Representación por hardware de splines

Mónica Jiménez Antón
Carlos Piñeiro Cordero
Cristina Valbuena Lledó

Profesor director:
Hortensia Mecha López

2006 – 2007
Proyecto de Sistemas Informáticos
Facultad de Informática
Universidad Complutense de Madrid

Resumen

El proyecto versa sobre la creación de un sistema de representación de imágenes tridimensionales basado en curvas polinómicas, en lugar de mallas de triángulos. Con este sistema, el coste de computación de la imagen generada depende en mayor medida del tamaño de la imagen que de la complejidad de la escena dibujada. Además, permite definir escenas con una mínima cantidad de datos, en comparación con las técnicas tradicionales.

Para ello, hemos realizado un desarrollo en tres fases. En primer lugar, investigamos sobre los fundamentos matemáticos necesarios para la generación de imágenes utilizando esta técnica. A continuación, implementamos la solución matemática mediante software. Y, por último, creamos un hardware básico capaz de demostrar que nuestro sistema era factible.

Abstract

The project deals with the creation of a three-dimensional images rendering system based on polynomial curves, instead of triangle meshes. With this system, the generated image's computational cost depends more on the image size than in the drawn scene's complexity. In addition, it allows to define scenes with a minimum amount of data, in comparison with the traditional techniques.

For that, we have done a three step development. In the first place, we investigated on the mathematical foundations needed for the generation of images using this technique. Next, we implemented on software the mathematical solution. And, finally, we created a basic hardware able to demonstrate that our system was feasible.

Lista de palabras

Spline, curva polinómica, malla, render, FPGA

Índice general

Índice de figuras

Capítulo 1

Introducción

Desde que se comenzó a utilizar la informática para la creación de imágenes sintéticas, las herramientas que se han puesto a disposición del diseñador han sido cada vez más y más sofisticadas y potentes. Sin embargo, aunque las formas han cambiado, el fondo se ha mantenido.

En la base de estas herramientas se encuentran las mallas de triángulos como estructura de datos. Toda herramienta de diseño tridimensional trabaja implícita o explícitamente con mallas como operandos y como producto. Así, una representación gráfica de un objeto tridimensional, ya sea un personaje de una película de animación o un coche de un videojuego, debe pasar por su conversión en malla de triángulos antes de dibujarse en una pantalla.

Esto se debe a que la utilización de mallas brinda una serie de ventajas con respecto a otros medios. Para empezar, los algoritmos que permiten proyectar tales mallas sobre una pantalla son relativamente sencillos y bien conocidos. Además, son muy paralelizables, lo que ha provocado en los últimos tiempos una especialización increíble de los procesadores gráficos.

Sin embargo, las mallas de triángulos siempre han tenido y tendrán una limitación muy grande: la incapacidad de representar superficies curvas. Nuestra intención es plantear un sistema de representación que se diferencia de los actuales desde la base, donde el límite no lo pone la definición de la malla, sino la definición de la pantalla sobre la que ésta se proyecta.

1.1. El proyecto

Nuestro proyecto consiste en la elaboración de un sistema de representación de superficies curvas no basado en la representación de polígonos. Concretamente, el tipo de objetos mostrados son *splines bicúbicas*, superficies matemáticas generadas a partir de una malla de control como la que se

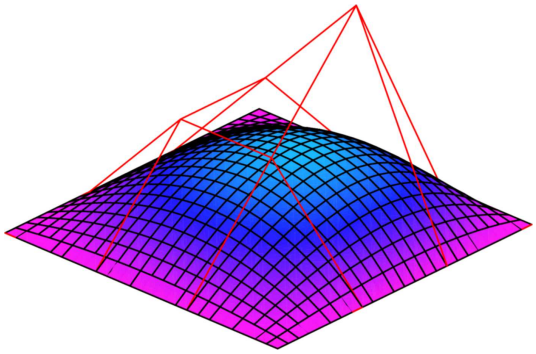


Figura 1.1: Spline bicúbica con su malla de control, en rojo.

puede ver en la figura 1.1. Este tipo de superficies matemáticas se formularon a mediados del siglo XX, y se han seguido utilizando hasta la actualidad.

Este trabajo consta de tres partes principales: desarrollo del algoritmo, desarrollo software y desarrollo hardware. Para el desarrollo del algoritmo hemos utilizado un programa matemático, Maple, que permite una productividad muy alta a la hora de trabajar con algoritmos matemáticos. Para la parte software, hemos utilizado C++ con el fin de conseguir que nuestro proyecto funcionase inicialmente en un ordenador personal. Finalmente, con el objetivo de mostrar que el sistema de representación que mostramos posee un nivel de paralelismo implícito muy grande, hemos desarrollado un hardware específico para la operación más complicada presente en el algoritmo, tratando de explotar al máximo dicho paralelismo.

1.1.1. Ventajas de la representación de splines

Si bien, como ya se ha dicho, la utilización de mallas es muy conveniente, sobre todo, por la facilidad con que se puede tratar con ellas, el método de representación desarrollado a lo largo de este proyecto provee de una serie de ventajas que el trabajo con mallas no puede proporcionar.

Para empezar, las mallas no pueden representar adecuadamente superficies curvas. Al estar formadas por una serie de polígonos planos, toda superficie curva se puede representar sólo mediante el uso de muchas superficies planas, con lo que el resultado conseguido no siempre es el esperado.

Con el fin de poder representar superficies curvas, lo que se suele hacer es aumentar la definición de las mallas utilizadas, por ejemplo, representando un círculo con un polígono regular de muchas caras. Sin embargo, esta solución tiene las siguientes desventajas:

- Si el observador se acerca a la superficie lo suficiente, puede apreciar

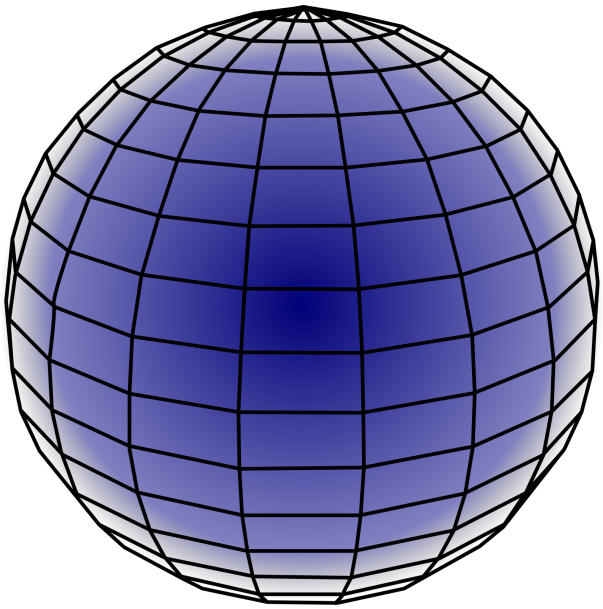


Figura 1.2: Malla de vértices de una esfera.

que, después de todo, la superficie sigue estando formada por polígonos planos.

- Si el observador se aleja de la superficie, ocurrirá que ésta tiene más definición de la que el observador puede apreciar, por lo que se estará trabajando innecesariamente al representar.
- El tráfico de datos aumenta: cuanto mayor es el número de vértices de la malla, mayor es la cantidad de datos a procesar.
- Las operaciones de rotación, translación y deformación de una malla compuesta por muchos vértices se vuelven más costosas: las transformaciones han de aplicarse sobre todos y cada uno de los vértices de la malla.

Supongamos, por ejemplo, el caso de una esfera. Si dividimos una esfera en 20 meridianos y 20 paralelos, y tomamos como vértices de una malla las intersecciones entre meridianos y paralelos, obtendremos una malla como la de la figura 1.2, con 380 vértices y 400 caras. Sin embargo, en el momento en que un observador se acerca a la esfera, como en la figura 1.3, fácilmente puede observar que está compuesta de planos. Por otro lado, cuando el observador se aleja suficientemente de la esfera, por muy pocos pixels que ocupe en la pantalla hay que seguir procesando todos los vértices, con lo que se realiza trabajo innecesario.

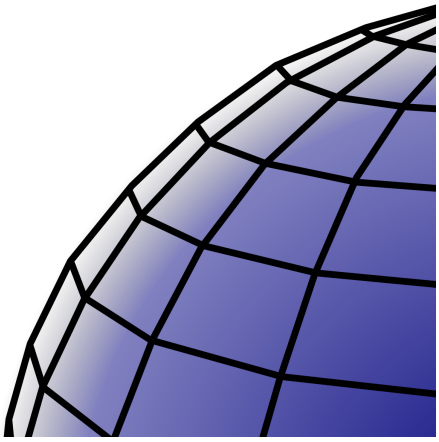


Figura 1.3: Detalle de la malla de la esfera. Vemos que no es en absoluto una esfera, sino un poliedro.

Si se utiliza el método de representación expuesto en este trabajo, se puede definir una esfera utilizando 8 superficies NURBS[1] compuestas, en total, de 128 vértices de control. Utilizando este sistema, por más que el observador se aproxime no podrá observar ningún defecto, pues se trata de una esfera definida mediante una expresión matemática en lugar de mediante una aproximación.

1.1.2. Objetivos del proyecto

El objetivo principal del proyecto es desarrollar un método de representación de superficies curvas en el que la cantidad de trabajo a realizar sólo dependa del tamaño de la imagen a mostrar. Concretamente, se trata de desarrollar un método de representación de splines bicúbicas, superficies matemáticas definidas por un polinomio cúbico en 2 variables. Nuestra intención es que el coste del algoritmo no dependa del número de vértices de la malla de control de las splines mostradas, como en el caso de las mallas de triángulos, sino que sólo dependa de la cantidad de la pantalla ocupada por

[1]Una superficie NURBS o *Non Uniform Rational B-Spline* es un tipo de spline bicúbica racional. Las splines racionales son un tipo de superficies que tienen una formulación matemática ligeramente distinta que las splines utilizadas en este proyecto pero que se pueden traducir a splines bicúbicas normales. *No uniformes* significa que dicho tipo de splines tienen un cuarto valor en sus vértices de control que se utiliza para ponderar la influencia que cada nodo tiene en la spline. Para aplicar nuestro algoritmo a este tipo de superficies, bastaría con generalizarlo para que pueda operar con vértices de 4 coordenadas, lo cual es trivial.

las splines. Para conseguirlo, el algoritmo tratará de calcular los puntos de corte entre las splines y las líneas de visión del observador, aproximándolos hasta un margen de error menor o igual que el tamaño de cada punto de la imagen. De este modo, no se trabajará con detalles más pequeños de lo que dicho observador pueda apreciar, al contrario de lo que ocurre al trabajar con mallas de triángulos.

La entrada de nuestro sistema es el conjunto de puntos de control que determinen la escena a representar, y la salida será la imagen representada en una pantalla.

Tras concebir un algoritmo que permita dicho trabajo, se pretende implementarlo en un lenguaje de programación de alto nivel como C++, con el fin de ser capaces de representar realmente en una pantalla splines iluminadas por un número de fuentes de luz personalizables por el usuario. Si bien por simplicidad realizaremos los cálculos, en la mayoría de las ocasiones, con una única spline, el algoritmo es totalmente capaz de operar con más de una spline simultáneamente.

Una vez conseguida una aplicación de este tipo, realizamos una aproximación hardware al problema, para así reducir el tiempo de ejecución en la medida de lo posible.

De esta forma, no sólo pretendemos demostrar que la representación de splines siguiendo este método es posible, sino que además, utilizando el hardware adecuado, se puede realizar de forma rápida. Además, demostramos que podemos representar superficies curvas con una mínima parte de los recursos necesarios utilizados para las mallas de triángulos, y que si tradujésemos nuestro algoritmo a una hipotética arquitectura CPU + GPU, como se trabaja en la actualidad, la cantidad de datos a enviar de CPU a GPU podría reducirse en órdenes de magnitud.

1.2. Organización del proyecto

Hemos dividido el proyecto en una serie de objetivos, de modo que entre una meta y la siguiente pudiésemos trabajar en paralelo, terminando cada meta poniendo todo en común. Dichas metas son:

1. Diseño matemático del algoritmo.

2. Implementación del algoritmo en un lenguaje matemático (Maple).

3. Implementación en un lenguaje de alto nivel (C++).

4. Diseño del hardware.

5. Implementación de hardware específico en VHDL.

6. Integración final hardware–software.

1.2.1. Diseño matemático del algoritmo

El primer problema al que nos enfrentamos fue que nuestros conocimientos inmediatos sólo abarcaban cómo calcular la intersección entre una recta y una curva en el plano, y necesitábamos generalizarlo a la intersección de una recta con una superficie en el espacio. Esto significa que, en el momento de comenzar, no sabíamos el tiempo que podía llevar resolver dicho problema, puesto que no hemos sido capaces de encontrar ningún ejemplo en la labor de documentación previa.

Una vez supiéramos cómo calcular el punto de intersección, tendríamos que averiguar cómo calcular la normal a la superficie en dicho punto para, finalmente, calcular la luz que el observador percibe en cada pixel.

1.2.2. Implementación en un lenguaje matemático

Tras conocer matemáticamente qué tipo de operaciones debíamos realizar, tuvimos que traducir dichas operaciones a un algoritmo que fuese implementable en términos de programación. Para ello, utilizamos Maple, un programa matemático que nos permite trabajar a muy alto nivel. Con un lenguaje que parece pseudo–código pero que tiene una gran cantidad de funciones gráficas, Maple nos permitió entender gráficamente el algoritmo al tiempo que lo implementábamos.

1.2.3. Implementación en un lenguaje de alto nivel

Cuando conseguimos traducir a un lenguaje sencillo las operaciones matemáticas, estructuramos todo el conocimiento obtenido en un programa que permitiera dibujar de forma sencilla splines. En esta etapa utilizamos C++, pues permite trabajar tanto a bajo como a alto nivel. Trabajar a alto nivel tiene la ventaja de mantener un cierto nivel de abstracción y dotar al código de una gran carga semántica. Por otro lado, podremos acceder al hardware de forma directa, a muy bajo nivel, sin sacrificar la semántica.

1.2.4. Diseño del hardware

Después de implementar en un lenguaje de alto nivel el algoritmo, optimizamos las partes que resultaron excesivamente lentas en software, explotando

el paralelismo implícito de las operaciones. A continuación, diseñamos una arquitectura específica para la operación que estaba llevando más tiempo en software aún con las optimizaciones.

1.2.5. Implementación en VHDL

Con el diseño en mano, lo implementamos en un lenguaje de descripción hardware, concretamente VHDL. La elección de este lenguaje en lugar de otros como Verilog, o del diseño esquemático, se debe, principalmente, a que los conocimientos que poseemos sobre el tema son mucho más amplios en VHDL que en cualquiera de sus alternativas.

1.2.6. Integración final hardware–software

Finalmente, realizamos una integración entre el software desarrollado en C++ y el hardware implementado en la fase anterior. Necesitamos diseñar e implementar algún tipo de interfaz hardware–software, de modo que el algoritmo se pudiera comunicar con el hardware que realizaba los cálculos correspondientes. Para ello, tuvimos que diseñar un protocolo tanto hardware como software, e implementarlo.

1.3. Herramientas utilizadas

1.3.1. Hardware

Durante el desarrollo del proyecto, utilizamos nuestros ordenadores personales, así como los recursos que la facultad puso a nuestra disposición. En concreto, utilizamos una placa *Xilinx Virtex II Pro*, que se compone de una FPGA, dos procesadores *PowerPC 405* y una serie de periféricos entre los que se incluye una memoria dinámica de 512MBytes y un conversor digital–analógico con salida VGA.

1.3.2. Software

El software utilizado es el siguiente:

- *Maple 8* para *Windows* y *Maple*[2] *10* para *Linux*.
- Entorno de desarrollo *KDevelop*[3] para *Linux* y *Microsoft Visual Studio*[4]

[2] `http://www.maplesoft.com`
[3] `http://www.kdevelop.org`
[4] `http://www.microsoft.com/spanish/msdn/latam/vstudio/default.aspx`

para *Windows*.

- Depurador *GDB*[5] del proyecto *GNU* para depurar la implementación en C++, y el conjunto de programas *Valgrind*[6] para trazar y corregir las pérdidas de memoria.
- GNU Profiler[7], utilidad para crear perfiles de ejecución de programas.
- *Xilinx ISE*[8] para el desarrollo del hardware en VHDL.
- *Xilinx ModelSim*[9] para la simulación del hardware implementado.
- *Xilinx Platform Studio*[10] para la integración hardware–software.
- *Eclipse*[11] como herramienta para la utilización de un CVS.
- LaTeX para la realización de esta memoria.

1.3.3. Almacenamiento del proyecto

Para almacenar los ficheros del proyecto hemos utilizado un CVS. El CVS[12], es una aplicación informática que implementa un sistema de control de versiones: mantiene el registro de todo el trabajo y los cambios en los ficheros que forman un proyecto, y permite que distintos desarrolladores colaboren. CVS nos pareció una buena alternativa debido a que nos permite consultar el estado del proyecto desde cualquier ordenador conectado a la red y recuperar versiones antiguas.

CVS utiliza una arquitectura cliente–servidor: un servidor guarda la versión actual del proyecto y su historial. Los clientes se conectan al servidor para obtener una copia completa (o parcial) del proyecto. Después, trabajan sobre dicha copia y, finalmente, ingresan sus cambios en el proyecto.

El servidor mantiene un registro de la historia de los cambios de los ficheros de un proyecto. De este modo, los desarrolladores pueden acceder no sólo a la última versión, sino a todas las versiones intermedias de un archivo. CVS permite que varios desarrolladores trabajen simultáneamente

[5] `http://sourceware.org/gdb/`
[6] `http://valgrind.org/`
[7] `http://www.gnu.org`
[8] `http://www.xilinx.com/ise/logic_design_prod/foundation.htm`
[9] `http://www.xilinx.com/ise/optional_prod/mxe.htm`
[10] `http://www.xilinx.com/ise/embedded_design_prod/platform_studio.htm`
[11] `http://www.eclipse.org`
[12] *Concurrent Versions System.*

sobre el mismo proyecto e, incluso, sobre los mismos archivos del proyecto. El funcionamiento, de forma general, es el siguiente:

1. Un desarrollador obtiene una copia local del proyecto.
2. Modifica el proyecto.
3. Ingresa los cambios que realizó en cada archivo:
   a) Si la versión de la copia del desarrollador es la misma que la del servidor, los nuevos archivos se insertan en el árbol de desarrollo sin problemas.
   b) Si la versión del servidor es posterior a la versión local, el desarrollador debe, manualmente, modificar su copia local y, finalmente, insertarla en el servidor.

De este modo, es difícil que un desarrollador tenga que realizar prácticamente ningún trabajo a la hora de poner en común sus modificaciones con el resto del equipo. Mediante esta herramienta, llevamos todo el control de cambios sin necesidad de establecer un protocolo de gestión.

Hemos utilizado *Eclipse* para trabajar con el CVS porque posee un interfaz para el trabajo en grupo muy fácil de utilizar y muy visual, con el que resulta muy sencillo mezclar el código cuando se producen conflictos.

Parte I

Desarrollo del algoritmo

Capítulo 2

Diseño del algoritmo principal

La idea inicial del algoritmo que desarrollaremos a continuación fue conseguir representar superficies curvas para que se vieran de la forma más realista posible. Lo primero fue diseñar un algoritmo que nos permitiera calcular la intersección de la superficie con cada línea de visión del observador. A la hora de implementar un algoritmo que haga esto de forma "matemática", descubrimos que los cálculos necesarios para averiguar el punto de corte entre una recta y una superficie curva son muy complejos y muy numerosos; además, en el caso de una imagen hay miles de líneas de visión a considerar, por lo que la tarea se torna inabordable.

Para superar esta dificultad, recurrimos a una simplificación de nuestro problema: calcular el punto de corte entre una recta y una curva Spline en el plano. Esta tarea se puede realizar con un algoritmo aproximativo, que calcula el punto de corte con un margen de error máximo especificado. En nuestro caso, nos bastaría hacer que el margen de error fuera tan pequeño como cada punto de la imagen. Por lo tanto, la solución que nosotros hemos conseguido pasa por generalizar el algoritmo a $\mathbb{R}^3$.

2.1. Referencias históricas

El concepto de *spline* proviene de la construcción naval en las épocas anteriores al Renacimiento, cuando no se realizaban diseños antes de construir los barcos. En lugar de utilizar matemáticas, los artesanos curvaban la madera hasta describir formas óptimas para la navegación, es decir, *splines*.

A principios del siglo XX, el diseño del fuselaje de los aviones se realizaba utilizando secciones cónicas: por primera vez, ciertos coeficientes cónicos podían utilizarse para definir formas y así los números sustituyeron a los dibujos. El uso de números en la descripción de una forma proporciona mayor

fidelidad al modelo original.

A finales de la década de 1950, apareció hardware para mecanizar la construcción de formas tridimensionales a partir de bloques de madera o acero. Estas formas podían utilizarse entonces como moldes para producir la chapa de los coches. El cuello de botella de este método de producción se reveló en la falta de software adecuado: para construir una forma utilizando un ordenador se hacía necesario producir una descripción de la misma compatible con un ordenador. Así, se pasó de las splines de madera a las splines por computador.

2.2. Nociones básicas

Los fundamentos teóricos expuestos a continuación han sido extraídos en parte del libro citado en la bibliografía [1] y en parte de los apuntes de la asignatura *Geometría Computacional*, impartida en la Facultad de Informática por Luis Pozo Coronado.

2.2.1. Spline

Una forma de definir una spline no lineal o curva elástica es como la curva de menor tensión posible definida por un polígono de control. El polígono de control es una sucesión de puntos; la spline pasa por el primero y por el último, mientras que los puntos intermedios afectan a su forma menos rigurosamente. Podemos ver una spline como una varilla metálica en la que los extremos se fijan y en ciertos puntos intermedios se tira de la misma con unos muelles; la varilla trazará la curva que pase más "cómodamente" entre el principio y el final, condicionada por la acción de los muelles, como se ve en la figura 2.1.

La expresión matemática explícita de una spline no lineal es muy complicada; por suerte, la aproximación lineal al problema variacional de minimización de la energía de deformación elástica son curvas polinómicas definidas a trozos de grado 3, que son lo que, generalmente, se entiende por splines. Al ser polinomios cúbicos a trozos, el tratamiento numérico por medio de un computador es un problema muy accesible.

2.2.2. Interpolación lineal

La interpolación lineal es una aplicación matemática que constituye la base del proceso de construcción de una spline. Nos permite conocer, a partir

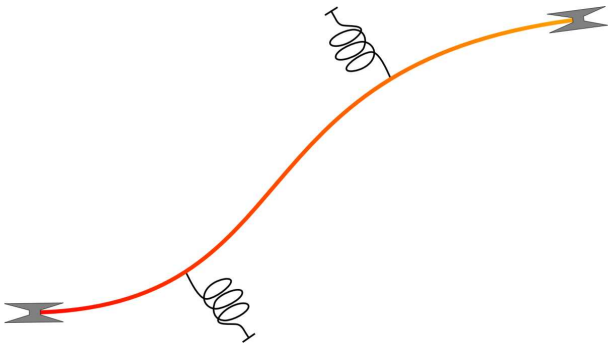


Figura 2.1: Varilla curvada por la acción de muelles.

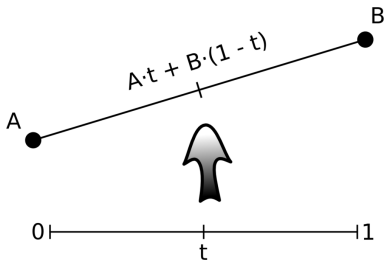


Figura 2.2: Interpolación lineal.

de dos puntos cualesquiera, todos los puntos que forman parte de la recta que pasa por los dos primeros.

Es decir, dados dos puntos a y b en $\mathbb{E}^3$, el conjunto de todos los puntos $x \in \mathbb{E}^3$ de la forma

$$x = x(t) = (1-t) \cdot a + t \cdot b \tag{2.1}$$

donde $t \in \mathbb{R}$ constituye la línea recta entre a y b. En el caso de $t = 0$, se cumple que $x(t) = a$; para $t = 1$ se cumple que $x(t) = b$ y, $\forall t \in (0, 1)$, el punto $x(t)$ está en el segmento que une a con b. Para cualquier otro valor de t, el punto está fuera del segmento $\overline{ab}$. En la figura 2.2 podemos ver cómo funciona la aplicación lineal.

La razón por la que las splines son una herramienta tan extendida en el diseño es porque se comportan "bien" geométricamente, en el sentido de que se comportan como uno espera. Es decir, si aplicamos una transformación afín al polinomio de control —por ejemplo, rotación, escala, translación, etc.—, los puntos de la spline que resultan de construir la curva a partir del polígono transformado son los mismos que se obtienen si primero se construye la spline y luego se aplica la transformación a sus puntos.

Esta propiedad se debe a que la interpolación lineal es una aplicación *invariante afín*: si ϕ es una transformación afín $\phi : \mathbb{E}^3 \to \mathbb{E}^3$, y la igualdad

(2.1) se cumple, entonces también se cumple

$$\phi x = \phi((1-t)\cdot a + t\cdot b) = (1-t)\cdot \phi a + t\cdot \phi b$$

La cuestión ahora es cómo construir la spline a partir de una serie de puntos utilizando la interpolación lineal. Paul de Casteljau, trabajando para Citroën, comenzó a trabajar en 1959 en un mecanismo para construir splines a partir de un polígono de control —también llamado polígono de Bézier, en honor a otro ingeniero que trabajó en el mismo campo—. Este algoritmo se basa en la idea de construir polinomios de grado k a partir de una sucesión de $k+1$ puntos, de forma que haya invarianza afín, mediante interpolación lineal iterada.

La interpolación lineal iterada consiste en, dada una sucesión de puntos $p_0, \ldots, p_n$ en $\mathbb{E}^2$ ó $\mathbb{E}^3$, y $t \in [0,1]$, obtener:

$$b_i^r(t) = (1-t)\cdot b_i^{r-1}(t) + t\cdot b_{i+1}^{r-1}(t)$$

donde $r \in \{1,\ldots,n\}$, $i \in \{0,\ldots,n-r\}$ y $b_i^0(t) = p_i$, $\forall i \in \{0,\ldots,n\}$. Entonces, $b_0^n(t)$ es el punto de la curva de Bézier b^n o spline de parámetro t. Así que $b^n(t) = b_0^n(t)$.

Ejemplo *Sean los puntos $p_0, p_1, p_2 \in \mathbb{E}^3$ y $t \in \mathbb{R}$, obtenemos la curva de Bézier que define el polígono formado por dichos puntos. Por interpolación lineal, sabemos que:*

$$\begin{aligned} b_0^1(t) &= (1-t)\cdot p_0 + t\cdot p_1 \\ b_1^1(t) &= (1-t)\cdot p_1 + t\cdot p_2 \end{aligned}$$

Sean ahora $b_0^1(t)$ y $b_1^1(t)$ los puntos que hemos obtenido, repetimos la interpolación lineal:

$$b_0^2(t) = (1-t)\cdot b_0^1(t) + t\cdot b_1^1(t)$$

Nótese que

$$b(t) = b_0^2(t) = (1-t)\cdot b_0^1(t) + t\cdot b_1^1(t) = (1-t)^2\cdot p_0 + 2\cdot t\cdot (1-t)\cdot p_1 + t^2\cdot p_2$$

Esto es un polinomio en t^2, así que $b_0^2(t)$ describe una parábola con $t \in (-\infty,\infty)$ que se denota por b^2. Su geometría se ilustra en la figura 2.3.

Los coeficientes intermedios se pueden escribir en una matriz triangular de puntos, llamada *esquema de de Casteljau*. Por ejemplo, en el caso cúbico,

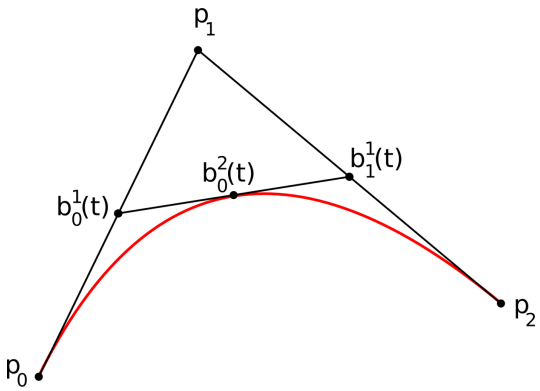


Figura 2.3: Parábola construida mediante interpolación lineal iterada.

la matriz queda de la siguiente manera:

$$\begin{array}{cccc} b_0 & & & \\ b_1 & b_0^1 & & \\ b_2 & b_1^1 & b_0^2 & \\ b_3 & b_2^1 & b_1^2 & b_0^3 \end{array}$$

donde, si los puntos originales son p_0, p_1, p_2, p_3, se cumple que

$$\begin{aligned} b_0 &= p_0 \\ b_1 &= p_1 \\ b_2 &= p_2 \\ b_3 &= p_3 \end{aligned}$$

2.2.3. Polinomios de Bernstein.

Las curvas de Bézier pueden ser definidas mediante un algoritmo recursivo, que es como de Casteljau las desarrolló por primera vez. Es también necesario, sin embargo, tener una representación explícita de ellas. Esto facilitará el desarrollo teórico sobre las curvas de forma considerable.

Los polinomios de Bernstein se definen explícitamente por:

$$B_i^n(t) = \binom{n}{i} \cdot t^i \cdot (1-t)^{n-i} \tag{2.2}$$

donde los coeficientes binomiales vienen dados por:

$$\binom{n}{i} = \begin{cases} \dfrac{n!}{i!(n-i)!} & \text{si } 0 \leq i \leq n \\ 0 & \text{en otro caso} \end{cases}$$

Una de las propiedades más importantes de los polinomios de Bernstein es que se pueden expresar de forma recursiva:

$$B_i^n(t) = (1-t) \cdot B_i^{n-1}(t) + t \cdot B_{i-1}^{n-1}(t)$$

con $B_0^0(t) = 1$ y $B_j^n(t) = 0$ para $j \notin \{0, \ldots, n\}$. Otra propiedad muy importante de los polinomios de Bernstein es que constituyen una *partición de la unidad*:

$$\sum_{j=0}^{n} B_j^n(t) = 1$$

Los puntos intermedios de de Casteljau b_i^r se pueden expresar en términos de los polinomios de Bernstein de grado r:

$$b_i^r(t) = \sum_{j=0}^{r} b_{i+j} \cdot B_j^r(t)$$

Esta ecuación muestra cómo los puntos intermedios b_i^r dependen directamente de los puntos de Bézier dados b_i. Utilizando los puntos intermedios b_i^r, podemos escribir una spline de la siguiente forma:

$$b^n(t) = \sum_{i=0}^{n-r} b_i^r(t) \cdot B_i^{n-r}(t) \tag{2.3}$$

No obstante, la expresión de Bernstein de una curva de Bézier, a partir de los puntos de control, es:

$$b(t) = \sum_{i=0}^{n} b_i \cdot B_i^n(t)$$

2.2.4. Blossom de una curva

El Blossom de una curva de Bézier $B(b_0, \ldots, b_n; \bullet)$ es la aplicación:

$$\begin{array}{rcl} B[b_0, \ldots, b_n; \bullet] : [0,1]^n & \rightarrow & \mathbb{E}^2 \text{ó } \mathbb{E}^3 \\ (t_1, \ldots, t_n) & \mapsto & b[b_0, \ldots, b_n; t_1, \ldots, t_n] \end{array}$$

donde $b[b_0, \ldots, b_n; t_1, \ldots, t_n] = b[t_1, \ldots, t_n] = b(t)$, construida del siguiente modo:

$$\begin{array}{l} b_i^0[t] = b_i \\ b_i^1[t_1] = (1-t_1) \cdot b_i + t_1 \cdot b_{i-1} \\ b_i^2[t_1, t_2] = (1-t_2) \cdot b_i^1[t_1] + t_2 \cdot b_{i+1}^1[t_1] \\ \vdots \\ b[t_1, \ldots, t_n] = b_0^n[t_1, \ldots, t_n] = (1-t_n) \cdot b_0^{n-1}[t_1, \ldots, t_{n-1}] \\ \qquad\qquad + t_n \cdot b_1^{n-1}[t_1, \ldots, t_{n-1}] \end{array}$$

La forma general de la expresión explícita del Blossom es la siguiente:

$$b[t_1,\dots,t_n]=\sum_{i=0}^{n} b_i\cdot\left(\sum_{\substack{\{\sigma_1,\dots,\sigma_i\}\subseteq\{1,\dots,n\}\\ \{\sigma_{i+1},\dots,\sigma_n\}=\{1,\dots,n\}\setminus\{\sigma_1,\dots,\sigma_i\}}} t_{\sigma_1}\dots t_{\sigma_i}(1-t_{\sigma_{i+1}})\dots(1-t_{\sigma_n})\right) \tag{2.4}$$

donde σ es una permutación ($\sigma\in S_n$) tal que $\sigma_1=\sigma(1),\dots,\sigma_i=\sigma(i)\in$ subconjuntos de i elementos. Así, el sumatorio interno es sobre los subconjuntos de i elementos de $\{1,\dots,n\}$, donde $\{\sigma_1,\dots,\sigma_i\}$ son esos i elementos y $\{\sigma_{i+1},\dots,\sigma_n\}$ son el conjunto complementario.

Sabemos que $b[t^{(n)}]=b(t)$, así que en el caso general podemos definir la curva de Bézier como una combinación baricéntrica del siguiente modo:

$$b(t)=b[t^{(n)}]=\sum_{i=0}^{n} b_i\cdot\left(\binom{n}{i}\cdot(1-t)^{n-1}\cdot t^i\right) \tag{2.5}$$

Ejemplo *Si tenemos una curva de Bézier definida por los puntos b_0, b_1, b_2 y b_3, su Blossom es:*

$$B(b_0,b_1,b_2,b_3;t)=b[t^{(3)}]=(1-t)^3\cdot b_0+3\cdot t\cdot(1-t)^2\cdot b_1+3\cdot t^2\cdot(1-t)\cdot b_2+t^3\cdot b_3$$

Existe un isomorfismo entre el espacio de curvas polinómicas de grado n sobre el intervalo $[0,1]$*:*

$$t:[0,1]^n\to\mathbb{R}^2 \text{ ó } \mathbb{R}^3$$

y el espacio de aplicaciones multiafines simétricas

$$\phi: b[t_1,\dots,t_n]\mapsto b(t)=b[t^{(n)}]$$

Por lo tanto, toda curva polinómica es una curva de Bézier con vértices de control dados por su Blossom asociado.

2.3. Subdivisión de splines

Como hemos visto, toda curva polinómica es una curva de Bézier, y toda curva de Bézier tiene asociados unos vértices de control. Por lo tanto, debe existir un método que permita subdividir una curva, es decir, calcular los puntos de control necesarios para describir un trozo de una curva de Bézier.

2.3.1. Subdivisión de una curva en el plano

Sea $b : [0,1] \to \mathbb{R}^3$ una curva de Bézier con vértices de control $b_0, \ldots, b_n$. Como es polinómica, puedo considerar que b está definida sobre $\mathbb{R}$. Así, sea el intervalo $[\alpha, \beta] \subseteq \mathbb{R}$, entonces

$$b|_{[\alpha,\beta]} : [\alpha, \beta] \to \mathbb{R}^3$$

es una curva polinómica y, por lo tanto, de Bézier. Su Blossom es $b[t_1, \ldots, t_n]$, pero definido sobre el intervalo $[\alpha, \beta]^n$. Los vértices de control de la curva restringida al intervalo $[\alpha, \beta]$, $b|_{[\alpha,\beta]}$, son, como se explica con más detalle en [1]:

$$c_i = b[\alpha^{(n-i)}, \beta^{(i)}]$$

Así, para el caso de dividir la curva en dos partes, tomamos los intervalos $[0, \frac{1}{2}]$ y $[\frac{1}{2}, 1]$. Calculamos los vértices de control asociados a la primera mitad, es decir, el intervalo $[\alpha, \beta] = [0, \frac{1}{2}]$:

$$c_i = b\left[0^{(n-i)}, \left(\frac{1}{2}\right)^{(i)}\right] = b_0^i\left(\frac{1}{2}\right)$$

Los vértices de control asociados a la segunda mitad, es decir, el intervalo $[\alpha, \beta] = [\frac{1}{2}, 1]$:

$$d_i = b\left[\left(\frac{1}{2}\right)^{(n-i)}, 1^{(i)}\right] = b_i^{n-i}\left(\frac{1}{2}\right)$$

Podemos ver con qué nodos del esquema de de Casteljau se corresponden c_i y d_i:

$$\begin{array}{llll}
b_0 = c_0 & & & \\
\vdots & b_0^1(1/2) = c_1 & & \\
\vdots & \vdots & \ddots & \\
\vdots & \vdots & \vdots & b_0^{n-1}(1/2) = c_{n-1} \\
b_n = d_n & b_{n-1}^1(1/2) = d_{n-1} & \cdots & b_1^{n-1}(1/2) = d_1 \qquad b_0^n(1/2) = c_n = d_0
\end{array}$$

En la figura 2.4 podemos ver el esquema anterior sobre una curva.

2.3.2. Subdivisión de una superficie en el espacio

Hasta ahora, todas las definiciones y los algoritmos que hemos visto se aplicaban sobre curvas, ya sea en el espacio o en el plano, pero sólo curvas. Sin embargo, el objetivo del proyecto es la representación de superficies, no de curvas.

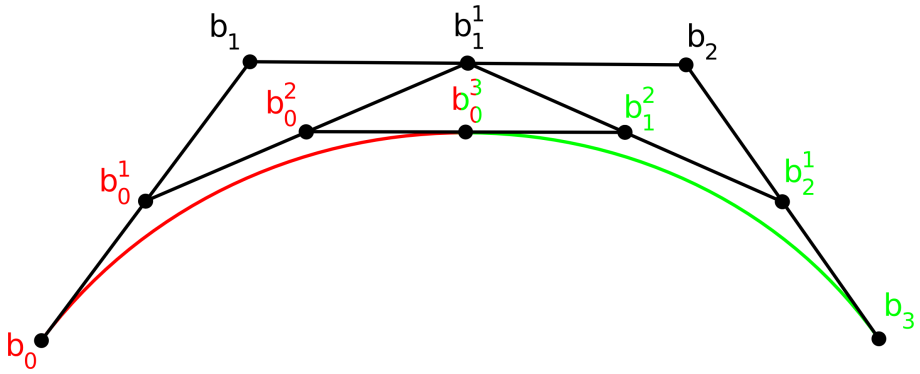


Figura 2.4: Obtención de los nodos de control de una sub-curva.

Interpolación bilineal

Mientras la interpolación lineal genera la *curva* más "simple" entre dos puntos, la interpolación bilineal encuentra la *superficie* más "simple" entre cuatro puntos.

Para ser más precisos: si tenemos cuatro puntos distintos

$$b_{0,0}, b_{0,1}, b_{1,0}, b_{1,1} \in \mathbb{E}^3$$

entonces el conjunto de todos los puntos x pertenecientes a $\mathbb{E}^3$ de la forma

$$x(u,v) = \sum_{i=0}^{1}\sum_{j=0}^{1} b_{i,j} \cdot B_i^1(u) \cdot B_j^1(v)$$

son un *paraboloide hiperbólico* que pasa por los cuatro puntos $b_{i,j}$.

Lo primero que llama la atención de la superficie es que, mientras la curva antes se podía expresar en función de un único parámetro t, ahora la interpolación bilineal se puede ver como una aplicación de la unidad cuadrada $0 \leq u, v \leq 1$ en el u,v-plano. Decimos que el plano unitario es el dominio del interpolante, mientras que la superficie x es su rango.

En lugar de evaluar el interpolante bilineal directamente, podemos aplicar un proceso de dos etapas mediante dos puntos intermedios:

$$\begin{aligned} b_{0,0}^{0,1} &= (1-v)\cdot b_{0,0} + v \cdot b_{0,1} \\ b_{1,0}^{0,1} &= (1-v)\cdot b_{1,0} + v \cdot b_{1,1} \end{aligned}$$

Obtenemos el resultado final como

$$x(u,v) = b_{0,0}^{1,1}(u,v) = (1-u)\cdot b_{0,0}^{0,1} + u \cdot b_{1,0}^{0,1} \tag{2.6}$$

Visto en la figura 2.5, sería como "avanzar" primero en u y después en v.

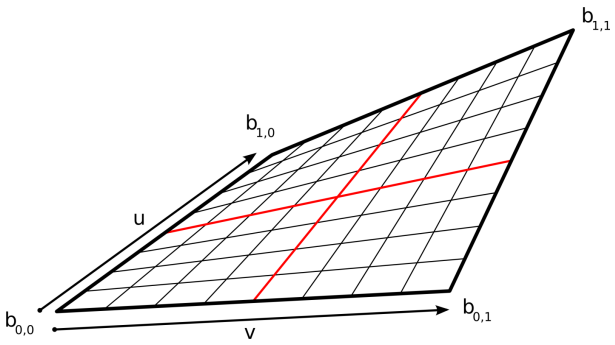


Figura 2.5: Interpolación bilineal: un paraboloide hiperbólico definido por cuatro puntos $b_{i,j}$.

Algoritmo de *de Casteljau* en superficies

Si la aplicación repetida de la interpolación lineal sobre una sucesión de puntos nos permitía obtener una curva de Bézier, la aplicación repetida de la interpolación bilineal proporcionará superficies.

Dada una matriz de puntos $b_{i,j}; 0 \leq i, j \leq n$, y los valores de los parámetros (u, v), el siguiente algoritmo genera un punto en una superficie determinada por la matriz de $b_{i,j}$:

Dados $\{b_{i,j}\}_{i,j=0}^{n}$ *y* $(u, v) \in \mathbb{R}^2$*, entonces:*

$$b_{i,j}^{r,r} = \begin{bmatrix} 1-u & u \end{bmatrix} \cdot \begin{bmatrix} b_{i,j}^{r-1,r-1} & b_{i,j+1}^{r-1,r-1} \\ b_{i+1,j}^{r-1,r-1} & b_{i+1,j+1}^{r-1,r-1} \end{bmatrix} \cdot \begin{bmatrix} 1-v \\ v \end{bmatrix}$$

donde $r = 1, \ldots, n$ *e* $i, j = 0, \ldots, n-r$ *y* $b_{i,j}^{0,0} = b_{i,j}$.

Entonces, $b_{0,0}^{n,n}(u, v)$ es el punto de parámetros (u, v) en la superficie de Bézier $b^{n,n}$.

Subdivisión

En los siguientes apartados, los conceptos matemáticos han sido desarrollados por nosotros, ya que la generalización a superficies no es algo trivial y no se encontraba en las fuentes consultadas.

Las superficies requeridas para nuestro propósito constan de una matriz de 4×4 puntos de control. Este número de puntos de control es interesante porque producen superficies que se describen en un polinomio bicúbico. Los polinomios bicúbicos son doblemente derivables y además su segunda derivada es continua, propiedades deseables en el diseño de superficies, puesto que

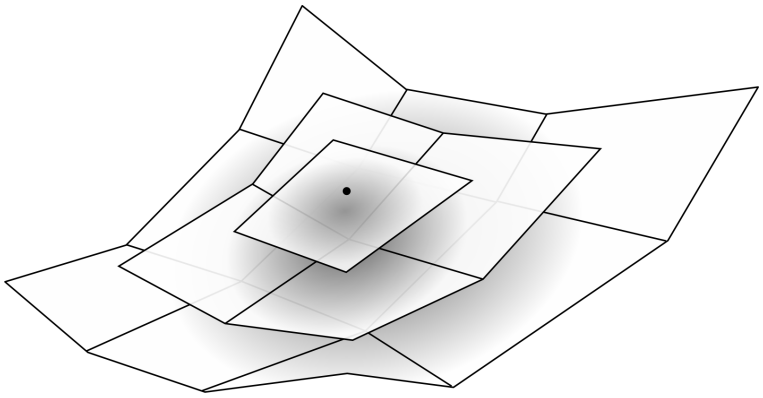


Figura 2.6: Aplicación del algoritmo de *de Casteljau* para superficies.

describen formas muy resistentes[1].

La subdivisión de superficies obtiene como resultado las matrices de puntos de control de las cuatro sub-superficies en que se divide una superficie para unos valores de u y v fijos. Lo que haremos será generalizar el proceso de división de curvas a superficies.

En el caso de las curvas, obteníamos los nodos de las sub-curvas a partir del Blossom de la curva. Sin embargo, resolver el problema del cálculo de los nodos de las sub-superficies a partir de Blossom es muy complejo matemáticamente.

La opción más sencilla es generalizar la relación que hay entre los nodos de las sub-curvas y los nodos intermedios del esquema de *de Casteljau* a superficies.

El problema es que el algoritmo de *de Casteljau*, que vemos en la figura 2.6, genera un esquema que no es una matriz triangular como en el caso de las curvas, sino una matriz tridimensional en forma de "pirámide rectangular". Igual que en el caso de las curvas los nodos de las sub-curvas se obtenían como el lado superior e inferior de la matriz, en el caso de superficies los nodos de las sub-superficies se corresponden con los nodos intermedios que están en los cuatro "lados" de la pirámide. Una vez más, el problema resulta bastante complejo.

Realizamos entonces un salto "creativo" para poder determinar qué nodos intermedios son nodos de control de las sub-superficies. Para ello, lo primero es resaltar algunas propiedades:

- La interpolación bilineal, como vimos en la ecuación (2.6), se podía

[1]Nos referimos a *mecánicamente* muy resistentes, razón por la cual se utilizan en el diseño, por ejemplo, de carrocerías de automóviles.

entender como interpolar primero en una "dirección" (v), y luego en la otra (u).

- En el caso de interpolar en u, a partir de los nodos $b_{i,j}^{n,m}$ y $b_{i+1,j}^{n,m}$ se interpola el nodo $b_{i,j}^{n+1,m}$. En el caso de v, a partir de $b_{i,j}^{n,m}$ y $b_{i,j+1}^{n,m}$ se obtiene el nodo $b_{i,j}^{n,m+1}$.
- El punto $b_{0,0}^{n,n}$ pertenece a las cuatro sub-superficies, al igual que ocurría con el punto b_0^n en el caso de las curvas.
- El borde de una superficie spline es una curva spline. Por lo tanto, al menos dos bordes de cada sub-superficie —los que comparten con la superficie original— se pueden calcular interpolando sólo en una "dirección" como si trabajásemos con curvas.
- Los nodos del borde de una sub-superficies debe coincidir con los nodos del borde adyacente de otra sub-superficie adyacente.

Teniendo todo esto presente, se pueden identificar qué nodos intermedios se corresponden con los nodos de control de las sub-superficies de una superficie 4×4. Aplicamos los conocimientos anteriores paso a paso:

1. Los bordes de una superficie spline son curvas spline; por lo tanto, la subdivisión del borde funciona como con las curvas:

$$\left(\begin{array}{ccc|c|ccc}
b_{0,3}^{0,0} & b_{0,3}^{1,0} & b_{0,3}^{2,0} & b_{0,3}^{3,0} & b_{1,3}^{2,0} & b_{2,3}^{1,0} & b_{3,3}^{0,0} \\
b_{0,2}^{0,1} & \star & \star & \star & \star & \star & b_{3,2}^{0,1} \\
b_{0,1}^{0,2} & \star & \star & \star & \star & \star & b_{3,1}^{0,2} \\
\hline
b_{0,0}^{0,3} & \star & \star & \star & \star & \star & b_{3,0}^{0,3} \\
\hline
b_{0,0}^{0,2} & \star & \star & \star & \star & \star & b_{3,0}^{0,2} \\
b_{0,0}^{0,1} & \star & \star & \star & \star & \star & b_{0,0}^{3,1} \\
b_{0,0}^{0,0} & b_{0,0}^{1,0} & b_{0,0}^{2,0} & b_{0,0}^{3,0} & b_{1,0}^{2,0} & b_{2,0}^{1,0} & b_{3,0}^{0,0}
\end{array}\right)$$

2. El punto $b_{0,0}^{3,3}$ pertenece a las cuatro sub-superficies y, además, es un nodo de control de las mismas:

$$\left(\begin{array}{ccc|c|ccc} b_{0,3}^{0,0} & b_{0,3}^{1,0} & b_{0,3}^{2,0} & b_{0,3}^{3,0} & b_{1,3}^{2,0} & b_{2,3}^{1,0} & b_{3,3}^{0,0} \\ b_{0,2}^{0,1} & \star & \star & \star & \star & \star & b_{3,2}^{0,1} \\ b_{0,1}^{0,2} & \star & \star & \star & \star & \star & b_{3,1}^{0,2} \\ \hline b_{0,0}^{0,3} & \star & \star & b_{3,3}^{0,0} & \star & \star & b_{3,0}^{0,3} \\ \hline b_{0,0}^{0,2} & \star & \star & \star & \star & \star & b_{3,0}^{0,2} \\ b_{0,0}^{0,1} & \star & \star & \star & \star & \star & b_{0,0}^{3,1} \\ b_{0,0}^{0,0} & b_{0,0}^{1,0} & b_{0,0}^{2,0} & b_{0,0}^{3,0} & b_{1,0}^{2,0} & b_{2,0}^{1,0} & b_{3,0}^{0,0} \end{array}\right)$$

3. Rellenamos el resto de la matriz contando con lo apuntado anteriormente sobre los bordes:

$$\left(\begin{array}{ccc|c|ccc} b_{0,3}^{0,0} & b_{0,3}^{1,0} & b_{0,3}^{2,0} & b_{0,3}^{3,0} & b_{1,3}^{2,0} & b_{2,3}^{1,0} & b_{3,3}^{0,0} \\ b_{0,2}^{0,1} & b_{0,2}^{1,1} & b_{0,2}^{2,1} & b_{0,2}^{3,1} & b_{1,2}^{2,1} & b_{2,2}^{1,1} & b_{3,2}^{0,1} \\ b_{0,1}^{0,2} & b_{0,1}^{1,2} & b_{0,1}^{2,2} & b_{0,1}^{3,2} & b_{1,1}^{2,2} & b_{2,1}^{1,2} & b_{3,1}^{0,2} \\ \hline b_{0,0}^{0,3} & b_{0,0}^{1,3} & b_{0,0}^{2,3} & b_{3,3}^{0,0} & b_{1,0}^{2,3} & b_{2,0}^{2,3} & b_{3,0}^{0,3} \\ \hline b_{0,0}^{0,2} & b_{0,0}^{1,2} & b_{0,0}^{2,2} & b_{0,0}^{3,2} & b_{1,0}^{2,2} & b_{2,0}^{1,2} & b_{3,0}^{0,2} \\ b_{0,0}^{0,1} & b_{0,0}^{1,0} & b_{0,0}^{2,1} & b_{0,0}^{3,1} & b_{1,0}^{2,1} & b_{2,0}^{1,1} & b_{3,0}^{3,1} \\ b_{0,0}^{0,0} & b_{0,0}^{1,0} & b_{0,0}^{2,0} & b_{0,0}^{3,0} & b_{1,0}^{2,0} & b_{2,0}^{1,0} & b_{3,0}^{0,0} \end{array}\right) \tag{2.7}$$

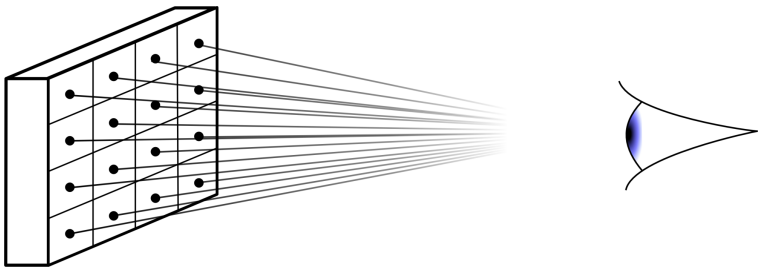


Figura 2.7: Representación del haz de rectas.

Y así hemos conseguido una matriz que nos proporciona los nodos de control de las cuatro sub-superficies obtenidas en la subdivisión de la superficie original. A partir de ellos ya podemos trabajar con cada superficie independientemente y seguir subdividiendo hasta que sea necesario.

2.3.3. Cálculo de un nodo intermedio

Finalmente, una vez supimos qué nodos necesitábamos para calcular las sub-mallas, sólo fue necesario hallar la fórmula explícita de los nodos intermedios:

$$b_{i,j}^{r,s} = \sum_{k=0}^{r} \sum_{l=0}^{s} b_{i+k,j+l} \cdot B_k^r \cdot B_l^s$$

donde:

- $b_{i,j}$ es el nodo (i, j) de la malla de control original.
- B_i^n es el polinomio de Bernstein i-ésimo de grado n evaluado en $t = 0{,}5$.

2.4. Representación por intersección

Nuestro siguiente objetivo es representar una superficie a partir de sus puntos de control. Dado que una superficie es un objeto en tres dimensiones y la representación se hace en la pantalla, necesitamos pasar de las tres dimensiones de la spline a las dos dimensiones de la pantalla. Para ello, calculamos los puntos de intersección entre la superficie y cada una de las líneas de visión. Podemos imaginar las líneas de visión como un haz de rectas que pasan por el ojo del observador y por cada uno de los pixels de la pantalla, como en la figura 2.7.

Para ello, consideramos que el foco de visión está situado en el punto $(0.5, 0.5, 2)$ y la pantalla forma un cuadrado vertical de lado 1 y de esquinas

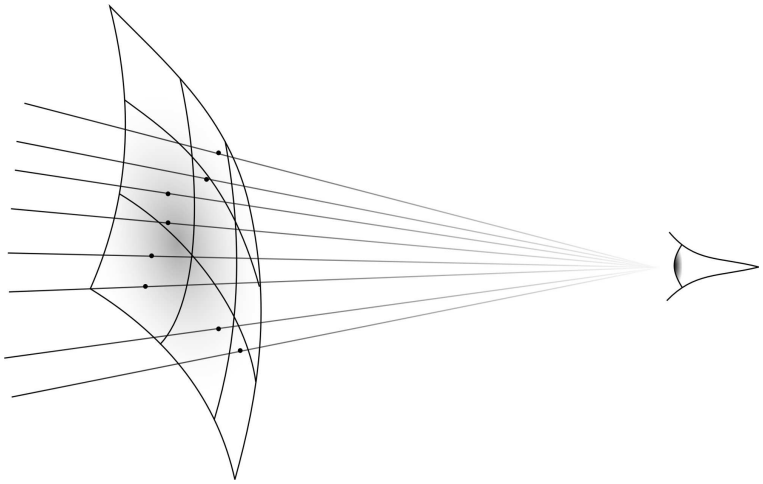


Figura 2.8: El algoritmo que nos ocupa calcula los puntos de intersección entre un haz de rectas y una superficie.

opuestas $(0,0,1)$ y $(1,1,1)$; dicho de otro modo, la escena se encuentra contenida en el cubo de esquinas opuestas $(0,0,0)$, $(1,1,1)$ y la pantalla está en el plano $z = 1$. Representamos la superficie según el campo de visión, es decir, calculamos la intersección de cada recta del foco con la superficie. Para cada recta, nos quedaremos con el punto de intersección que tenga un mayor valor de la coordenada z, puesto que será lo más cercano al foco de visión y, por tanto, lo que se vea. Después, calcularemos la normal a la superficie en dicho punto, obtendremos el color que un observador vería y, finalmente, representaremos ese color en una pantalla. Esquemáticamente, consiste en calcular todos los puntos de corte de las rectas de la figura 2.8 con la superficie que aparece.

2.4.1. Intersección de una recta con una curva de Bézier: algoritmo de intersección por división

El cálculo de la intersección de una recta con una curva de Bézier requiere, en primer lugar, encontrar la menor caja que contenga al polígono de control de la curva; la curva estará contenida en la caja por el hecho de estar contenida en la envoltura convexa del polígono. Para ello, se calcula el menor rectángulo $[x_0, x_1] \times [y_0, y_1]$ que contiene al conjunto de vértices de control de la curva de Bézier.

Una vez obtenida la caja, se aplica el *algoritmo de intersección por división*. Este algoritmo consiste en la comprobación del corte de la recta con cajas cada vez más pequeñas. Cuando la recta corta a una caja, se subdivide

la curva contenida en dicha caja mediante el algoritmo de *de Casteljau* visto anteriormente —el cálculo de los nodos intermedios que se corresponden con los c_i y d_i—. Recursivamente, se realiza la comprobación con cada una de las cajas que contienen las curvas resultantes.

En general, el algoritmo de intersección por división utiliza el valor del parámetro $t = 1/2$ para la subdivisión, aunque es un valor arbitrario que está comprobado que funciona bien. El algoritmo divide hasta la precisión deseada, que dará como resultado una caja lo suficientemente pequeña para que se pueda aproximar el punto de corte como su centro.

2.4.2. Intersección de una recta con una superficie

El cálculo de la intersección de una recta con una superficie es una generalización del caso anterior. Lo primero también es encontrar la caja mínima que contiene a la superficie. A continuación, se comprueba el corte de la recta con la caja y en caso de corte se subdivide.

La subdivisión ahora también se hace recurriendo al algoritmo de *de Casteljau*, generalizado para superficies: utilizando los valores de los parámetros $u = 1/2$ y $v = 1/2$, calculamos los nodos intermedios que vimos en la matriz (2.7). Así, la superficie queda dividida en cuatro submallas, en las que se comprueba recursivamente el corte con la recta.

Por último, el algoritmo devuelve una lista de puntos de corte. Como necesitamos saber cuál es el más cercano al observador, recurrimos al uso de un buffer Z para obtener el punto más cercano de todos.

Buffer Z

El algoritmo del Buffer Z sirve para, dado un conjunto de objetos, ordenarlos en función de su distancia al observador. En nuestro caso, lo único que se necesita es conocer el punto más cercano al observador de todos los que el algoritmo recibe como entrada. Para ello, aplicamos la "fuerza bruta": comparamos todos los puntos y nos quedamos con el de menor z.

Este sistema, que puede parecer poco sofisticado, es en realidad el mismo que se aplica en las tarjetas gráficas y, aunque parece muy rudimentario, produce los resultados muy rápidamente.

Corte de una recta y una caja

Aunque no lo hayamos tratado hasta ahora, el problema de decidir si las rectas cortan a las cajas que contienen las mallas de control de las splines no es un problema trivial. Se podría realizar una implementación conceptualmente

simple que realizase un gran número de comparaciones para determinar si una recta atraviesa una caja; sin embargo, si tenemos en cuenta que este procedimiento se ejecutará incluso más veces que la división de las mallas, merece la pena dedicar cierto a tiempo a este problema.

Lo primero que es necesario resolver es la representación tanto de la caja como de la recta. La caja se define por 6 planos, así que se podría utilizar simplemente una representación que tomase dos esquinas opuestas de la caja. De esta manera, dada la caja de esquinas $(x_0, y_0, z_0), (x_1, y_1, z_1)$ quedará definida por los planos $x = x_0$, $x = x_1$, $y = y_0$, $y = y_1$, $z = z_0$ y $z = z_1$. En cuanto a la recta, utilizaremos su definición paramétrica:

$$r \equiv (a \cdot \lambda + b, c \cdot \lambda + d, e \cdot \lambda + f)$$

Mediante esta representación, nuestra primera implementación fue bastante directa aunque no totalmente eficiente:

- Obteníamos la coordenada x menor (numéricamente) de la caja, que es x_0.
- Calculábamos el parámetro λ para que la recta tuviese ese valor en su coordenada x:
$$\lambda = \frac{x_0 - b}{a}$$
- Sustituíamos el parámetro λ para obtener las coordenadas en y y z de la recta:
$$y = c \cdot \lambda + d, \quad z = e \cdot \lambda + f$$
- Comprobábamos si y estaba entre las coordenadas y máxima y mínima de la caja (numéricamente), es decir, entre y_0 e y_1. En caso afirmativo, comprobábamos también con z.

Este procedimiento tan sencillo se repetía del siguiente modo:

- Obteníamos λ_{x_0} como hemos explicado, y comprobábamos que $y \in [y_0, y_1] \wedge z \in [z_0, z_1]$. Si se cumple la condición anterior, la recta corta a la caja. Si no es así, la recta no corta a la caja cuando pasa por x_0.
- Si la condición anterior no se cumplía, realizábamos el mismo cálculo pero obteniendo λ_{x_1}.
- Si la condición anterior tampoco se cumplía, comprobábamos con λ_{y_0}.
- ...

- Si la condición anterior tampoco se cumplía, comprobábamos con λ_{z_1}.

De este modo, si λ_{x_0} hacía que se cumpliera que $y \in [y_0, y_1] \wedge z \in [z_0, z_1]$, podíamos confirmar que la recta cortaba a la caja. Si, por el contrario, la recta no la cortaba, debíamos realizar otras 6 comprobaciones para λ_{y_0} y, en el peor caso, también para λ_{z_1}. Cada una de estas comprobaciones suponía:

- Despejar el parámetro λ: 1 resta y una división en punto flotante.
- Sustituir en las otras dos coordenadas: 2 productos y 2 sumas en punto flotante.
- Realizar dos comparaciones y una conjunción lógica: 2 comparaciones en punto flotante y una *and* bit a bit.

En total, suponen 6 divisiones en punto flotante, 12 productos, 18 sumas y restas, 12 comparaciones y 6 operaciones *and*. Aunque, en un principio, pudiera parecer que no es un cálculo muy grande, hay que tener en cuenta que este algoritmo es el que más se repite a lo largo del cálculo de la imagen, (puesto que antes de dividir hay que asegurarse de que la recta corta a la caja y, por lo tanto, se realiza más veces incluso que el algoritmo de subdivisión).

Optimización del corte de una recta con una caja

Teniendo en cuenta lo anterior, buscamos un método más óptimo. Para ello, redujimos el problema a la comprobación de si una recta corta a un rectángulo en el plano.

Sea la caja formada por la intersección de las siguientes rectas:

$$\begin{aligned} y &= y_0 \\ y &= y_1 \\ x &= x_0 \\ x &= x_1 \end{aligned}$$

entonces, si la recta está en forma paramétrica, siendo el parámetro λ, la forma de ver que corta a la caja es que pasa simultáneamente por la región del plano comprendida entre las rectas $y = y_0$ y $y = y_1$, y la región del plano comprendida entre $x = x_0$ y $x = x_1$. Los valores de λ para la intersección de las rectas que determinan el rectángulo con la recta que estamos tratando son:

- Para la recta $y = y_0$, el valor de λ es: λ_{y_0}.
- Para la recta $y = y_1$, el valor de λ es: λ_{y_1}.

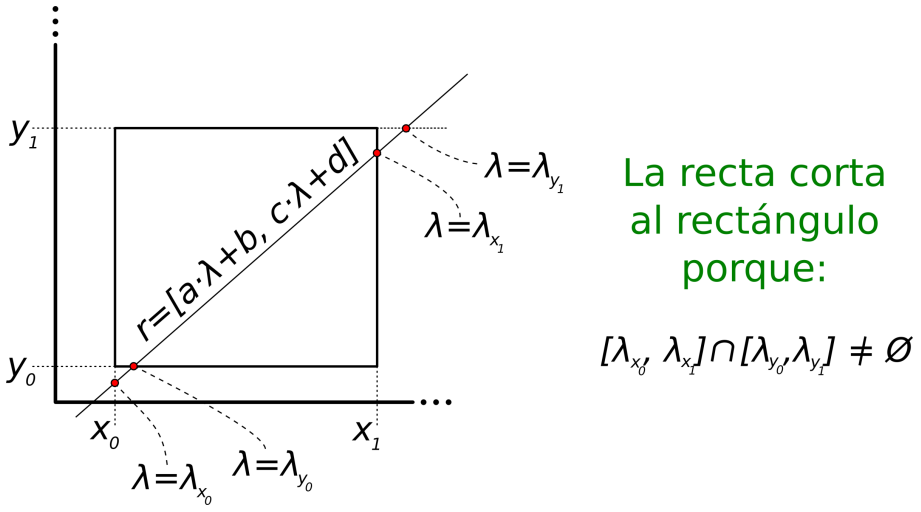


Figura 2.9: Intersección de una recta con un rectángulo.

- Para la recta $x = x_0$, el valor de λ es: λ_{x_0}.
- Para la recta $x = x_1$, el valor de λ es: λ_{x_1}.

Para determinar que la recta corta al rectángulo basta ver que los intervalos $[\lambda_{x_0}, \lambda_{y_0}]$ y $[\lambda_{x_1}, \lambda_{y_1}]$ se solapan, esto es que se cumpla que:

$$MAX(\lambda_{x_0}, \lambda_{y_0}) < MIN(\lambda_{x_1}, \lambda_{y_1})$$

La explicación queda representada en la figura 2.9.

Para el caso de la caja, la idea es sólo un poco más compleja. De hecho, la recta cortará a la caja si se cumple la siguiente desigualdad:

$$MAX(\lambda_{x_0}, \lambda_{y_0}, \lambda_{z_0}) < MIN(\lambda_{x_1}, \lambda_{y_1}, \lambda_{z_1})$$

De esta forma, se reducen notablemente las multiplicaciones y las sumas. Sin embargo, queda pendiente el problema de obtener las λs. Afortunadamente, sabiendo cómo se genera el haz de rectas, éstas se pueden obtener con pocos cálculos si se realizan adecuadamente.

El haz de rectas se genera teniendo en cuenta las siguientes observaciones:

- El observador se encuentra en la posición $(0.5, 0.5, 2)$.
- Para $\lambda = 0$, las rectas pasan por el observador.
- La "pantalla" —el plano de proyección— se encuentra en el plano $z = 1$. Para $\lambda = 1$, las rectas pasan por el plano de proyección.

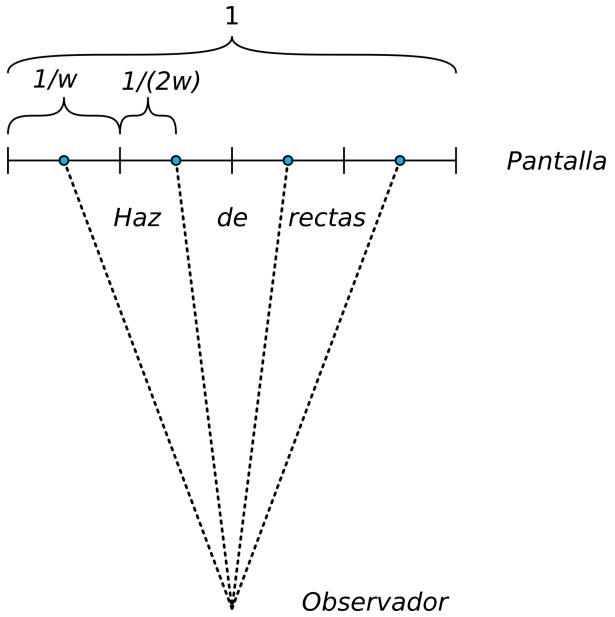


Figura 2.10: Distribución horizontal de un haz de rectas de anchura 4.

- Las rectas se distribuyen por el plano de proyección pasando por el centro de cada una de los rectángulos en que éste se divide. Podemos ver cómo se distribuyen horizontalmente en la figura 2.10.

Por lo tanto, sea una pantalla de w puntos de ancho y h puntos de alto, el conjunto de rectas que generan la imagen es:

$$\left\{\left[\left(\frac{i}{w}+\frac{1}{2\cdot w}-\frac{1}{2}\right)\cdot\lambda+\frac{1}{2},\left(\frac{j}{h}+\frac{1}{2\cdot h}-\frac{1}{2}\right)\cdot\lambda+\frac{1}{2},2-\lambda\right]\right\}_{i,j=0}^{i=w-1,j=h-1}$$

Así pues, conociendo cómo es el haz de rectas, podemos obtener los valores de λ de la siguiente manera:

- Para despejar λ_{z_0} y λ_{z_1}, sólo hay que realizar la operación $\lambda_z = z - 2$ dos veces.

- Para el caso de λ_{x_0}, λ_{x_1}, λ_{y_0} y λ_{y_1}, hay que despejar igual que antes (2 restas y dos divisiones en cada caso).

Finalmente, hay que realizar, en total, 5 comparaciones, con lo que el número de operaciones asciende a 6 restas en punto flotante, 4 divisiones y 5 comparaciones. Como vemos, el número de operaciones total se reduce enormemente.

2.5. Cálculo de la normal a la superficie

El vector normal a una superficie en un punto es el perpendicular a la superficie en dicho punto y de módulo 1.

Para los futuros cálculos de percepción del observador, es esencial poder calcular la normal a la superficie en el punto de intersección con las líneas de visión. Dicho punto no lo conocemos; nuestro algoritmo divide la spline en superficies más pequeñas para aproximarlo. El resultado son superficies diminutas de las que necesitamos calcular su normal. Hay dos opciones posibles:

- Calcular la normal en el punto medio de la superficie.
- Calcular la normal por métodos aproximativos.

El problema de calcular el vector normal a una superficie spline es que requiere una enorme cantidad de cálculos muy complejos. Por otro lado, el problema de utilizar un cálculo aproximativo es precisamente que nos permite conocer la normal a la superficie sólo de forma *aproximada*.

Para las comprobaciones en Maple[2], hemos implementado ambos métodos y se ha hecho un análisis comparativo con el que hemos llegado a la conclusión de que la normal *aproximada* se puede considerar igual que la normal "verdadera". Al tratarse de una superficie muy pequeña, ésta es casi plana y la normal es aproximadamente la misma en todos sus puntos.

2.5.1. Cálculo de la normal en un punto

El vector normal se puede obtener mediante el producto vectorial de dos vectores tangentes. Esto es:

$$N(u,v) = \frac{\frac{\partial}{\partial u}S_{n,m}(u,v) \times \frac{\partial}{\partial v}S_{n,m}(u,v)}{\left|\frac{\partial}{\partial u}S_{n,m}(u,v) \times \frac{\partial}{\partial v}S_{n,m}(u,v)\right|}$$

Donde $u, v \in [0,1]$, y para calcular la normal en el punto medio de la superficie, que es lo que nos interesa, les damos valor 0,5 a ambos.

Los vectores tangentes los obtenemos mediante las derivadas parciales de la superficie de Bézier en u y v. En la formula 2.8, el término entre corchetes es una curva de Bézier, por lo que su derivada se puede presentar como 2.9, quedando expresada como una curva de Bézier en la que los puntos de control son sustituidos por vectores dados por las diferencias de puntos de control consecutivos.

[2]Ver capítulo 3.2.9.

$$\frac{\partial}{\partial u}S_{n,m}(u,v) = \sum_{j=0}^{m}\frac{\partial}{\partial u}\left[\sum_{i=0}^{n}P_{i,j}B_i^n(u)\right]B_j^m(v) \tag{2.8}$$

$$\frac{\partial}{\partial u}\left[\sum_{i=0}^{n}P_{i,j}B_i^n(u)\right] = n\sum_{i=0}^{n-1}(P_{i+1,j}-P_{i,j})B_i^{n-1}(u) \tag{2.9}$$

Para la derivada parcial respecto de v se opera de la misma forma. En resumen, las derivadas parciales de una superficie de Bézier para u y v son:

$$\frac{\partial}{\partial u}S_{n,m}(u,v) = n\sum_{j=0}^{m}\sum_{i=0}^{n-1}(P_{i+1,j}-P_{i,j})B_i^{n-1}(u)B_j^m(v)$$

$$\frac{\partial}{\partial v}S_{n,m}(u,v) = n\sum_{i=0}^{n}\sum_{j=0}^{m-1}(P_{i,j+1}-P_{i,j})B_j^{m-1}(v)B_i^n(u)$$

La derivada en cada dirección u y v sigue la dirección del vector tangente que va de los puntos de control del extremo de la superficie a los puntos de control vecinos. Por ello, el producto vectorial entre ambos devuelve un vector perpendicular a ambos, es decir, perpendicular a la superficie.

2.5.2. Cálculo de la normal por métodos aproximativos

El cálculo de la normal de manera aproximada se basa en dividir la malla en dos triángulos[3] y calcular la normal de cada uno de ellos. La normal de la malla se obtiene como el promedio de ambas.

Para calcular la normal de un triángulo nos basamos en que tres puntos no alineados determinan de forma única un plano. Sean los puntos A, B y C los puntos que determinan el triángulo, obtenemos los vectores $u = \overrightarrow{AB}$ y $v = \overrightarrow{AC}$ y a partir de ellos la normal como $\overrightarrow{n} = \overrightarrow{u} \times \overrightarrow{v}$, es decir, el vector $\overrightarrow{n}$ perpendicular a $\overrightarrow{u}$ y $\overrightarrow{v}$.

2.5.3. Estudio estadístico

El estudio estadístico realizado tuvo como objetivo la demostración de la corrección del cálculo de las normales mediante métodos aproximativos. La utilización de este método en lugar del cálculo de la normal en un punto reportaba una disminución considerable de los cálculos y de su complejidad.

[3]Para obtener los triángulos, utilizamos 3 esquinas de la malla. Sea la malla de nodos $\{b_{ij}\}_{i=0,j=0}^{3,3}$, los triángulos serían $b_{00}, \widehat{b_{30}, b_{33}}$ y $b_{33}, \widehat{b_{03}, b_{00}}$.

Para realizarlo, lo primero fue obtener una muestra. Nos interesaban las diferencias de ángulos entre las normales calculadas sobre las mismas mallas mediante ambos métodos, así que las utilizamos como individuos de la población de la que extrajimos la muestra.

A partir de la muestra, realizamos el análisis descriptivo de la misma:

1. Cálculo de la media. La media nos permite determinar alrededor de qué valor se agrupan los datos. En nuestro caso, nos interesaba que fuera un valor próximo a cero, lo que significaría que todas las diferencias de ángulos están alrededor del cero.

$$\overline{x} = \frac{\sum_{i=1}^{n} x_i}{n}$$

2. Cálculo de la varianza. La varianza nos permite medir la dispersión de los datos, que en nuestro caso nos interesa que sea lo menor posible.

$$s^2 = \frac{\sum_{i=1}^{n} (x_i - \overline{x})^2}{n-1}$$

3. Cálculo de la desviación típica. La desviación típica es otra medida que expresa la dispersión de la distribución.

$$s = \sqrt{s^2}$$

En resumen, si todos los datos fueran iguales (ángulos entre las normales cero), estas medidas también serían cero. Aunque en nuestro caso no son exactamente cero, sí se pueden aproximar a él y, por tanto, utilizar el cálculo de normales de forma aproximada con un error despreciable. Los datos obtenidos se pueden consultar en la sección 3.2.9.

2.6. Coloreado de los pixels: cálculo de la percepción del usuario

La siguiente información relativa a la iluminación se puede encontrar en [3].

En una escena, es necesario definir algún tipo de luz para calcular la percepción que tendría el usuario de cada punto. Si no hubiera luces, la escena se vería completamente negra.

2.6.1. Modelo de reflexión de *Phong*

Además de definir las luces, tuvimos que elegir un modelo de iluminación, es decir, la serie de reglas que dictan cómo se debe calcular la iluminación de la superficie. Nuestro modelo es una implementación parcial del modelo de *Phong*[4].

Este modelo de iluminación no se basa en una descripción física de la forma en que la luz ilumina superficies, sino que se trata de un modelo empírico que está demostrado que produce unos resultados bastante buenos. Phong define la cantidad de luz que percibe un observador como:

$$I = ambiental + \sum_{luces} (difusa + especular)$$

donde:

- I es la cantidad de luz que el observador percibe en un pixel.
- La luz ambiental es un valor de luz que se suma a todos los puntos de la escena. Se puede entender como una luz que viene de todas partes e ilumina todos los puntos de la escena por igual.
- La iluminación difusa es el tipo de iluminación que presentan todos los objetos sin brillo.
- La iluminación especular se refiere al tipo de brillos que aparecen en las superficies bruñidas.

En nuestro caso, no utilizaremos la luz ambiental —la cual se podría implementar muy fácilmente—, utilizando sólo la luz difusa y la especular.

Iluminación difusa

La iluminación difusa descrita en la tesis doctoral de Phong se basa en el modelo de reflexión lambertiano[5], el cual describe la cantidad de luz que un observador percibe mediante 3 leyes:

[4] *Bui Thong Phong* (vietnamita, 1942 – 1975) fue un investigador y pionero de los gráficos generados por computador. Fue el inventor del *modelo de reflexión Phong*, una técnica muy utilizada en la generación de imágenes por computador. Su tesis doctoral versó sobre este modelo de reflexión, y recibió el apoyo de David C. Evans e Ivan Sutherland, fundadores de la mítica *Evans & Sutherland* (`www.es.com`), una de las primeras empresas dedicadas a la creación de imagen sintética.

[5] Johann Heinrich Lambert, 1728 – 1777. Matemático, físico y astrónomo alemán nacido en Mülhausen (actualmente Francia), pionero en el estudio de la iluminación.

Primera ley de Lambert La iluminación de una superficie iluminada por la luz que cae sobre ella perpendicularmente desde una fuente de luz, es proporcional a la inversa del cuadrado de la distancia entre la superficie y la fuente de luz.

Segunda ley de Lambert Si los rayos de la fuente de luz alcanzan la superficie en ángulo, entonces la iluminación es proporcional al coseno del ángulo que forman con la normal a la superficie.

Tercera ley de Lambert La intensidad luminosa de la luz decrece exponencialmente con la distancia al viajar a través de un medio con absorción, esto es, un medio distinto del vacío.

Aplicando estas leyes, Phong describe la iluminación difusa que produce una fuente de luz como

$$I_d = k_d \cdot i_d \cdot (\overrightarrow{L} \bullet \overrightarrow{N})$$

donde:

- k_d es la constante de reflexión difusa de la superficie, y es dependiente del material de ésta.
- i_d es la intensidad de la fuente luminosa en difusa. Se trata de un valor dependiente de la iluminación.
- $\overrightarrow{L}$ es el vector que va desde la superficie hasta la fuente de luz.
- $\overrightarrow{N}$ es un vector normal a la superficie.
- $\overrightarrow{L} \bullet \overrightarrow{N}$ es el producto escalar de ambos vectores. Si son unitarios, dicho producto es, además, el coseno del ángulo que forman entre sí.

Básicamente, Phong describe la iluminación difusa sólo como la segunda ley de Lambert. En cuanto a la primera y la tercera, aunque se pasen por alto, su implementación no es excesivamente costosa.

Iluminación especular

La iluminación especular, según la define Phong, se puede entender como la cantidad de luz reflejada del foco que percibe el usuario. Imaginemos una lámpara que ilumina una madera barnizada: la luz difusa es la que nos permite ver que toda la madera es marrón, mientras que la especular es el brillo o reflejo que la lámpara produce en la madera.

La cantidad de luz así definida se calcula mediante la siguiente fórmula:

$$I_s = k_s \cdot i_s \cdot (\overrightarrow{R} \bullet \overrightarrow{V})^{\alpha}$$

donde:

- k_s es la constante de reflexión del material de la superficie.
- i_s es la intensidad de luz especular que produce la fuente luminosa.
- $\overrightarrow{V}$ es el vector que va desde la superficie hasta el observador.
- $\overrightarrow{R}$ es el vector reflejado de $\overrightarrow{L}$ respecto de la normal a la superficie, $\overrightarrow{N}$, donde $\overrightarrow{L}$ es el vector que une la superficie con la fuente de luz.
- $\overrightarrow{R} \bullet \overrightarrow{V}$ es el producto escalar de $\overrightarrow{R}$ y $\overrightarrow{V}$; además, si ambos vectores son unitarios, es el coseno del ángulo que forman.
- α es la constante de brillo, que nos permite distinguir entre, por ejemplo, el tipo de brillo que aparece en una bola de plástico y otra de aluminio. En nuestro caso, para reducir la complejidad del algoritmo, hemos cambiado el factor exponencial α por una función lineal, determinada empíricamente, que también produce buenos resultados.

Dicho de otro modo, la cantidad de luz especular es menor cuanto mayor es el ángulo que forman el observador y el rayo de luz reflejado por la superficie. Cuanto más directamente llegue el rayo reflejado al observador, mayor será el brillo que éste percibirá.

Para calcular el vector reflejado por la superficie, aplicaremos la siguiente fórmula:

$$\overrightarrow{R} = 2 \cdot \overrightarrow{N} \cdot (\overrightarrow{V} \bullet \overrightarrow{N}) - \overrightarrow{V}$$

donde:

- $\overrightarrow{N}$ es el vector normal a la superficie que sirve de espejo.
- $\overrightarrow{V}$ es el vector que incide en la superficie.
- $\overrightarrow{R}$ es el vector reflejado.

La intensidad de luz de la fuente es un parámetro que fija el usuario, así como los coeficientes de reflexión especular y difuso, que dependen del tipo de material del objeto. Para nuestro trabajo, el tipo de material utilizado es muy simple: se trata de un material blanco que refleja todos los colores

que inciden sobre él, y tiene un brillo similar al del metal. Aunque hemos realizado esta simplificación por comodidad, las modificaciones necesarias para poder representar distintos materiales son mínimas.

Una vez seamos capaces de obtener el color que el observador percibe en un punto de la imagen, sólo hay que aplicar el algoritmo para cada punto de la imagen hasta obtener la representación completa.

Capítulo 3

Implementación en Maple

3.1. Introducción

En la etapa inicial del proyecto realizamos los desarrollos matemáticos. Tras conseguir una formulación matemática del algoritmo de representación de splines, comprobamos su corrección mediante una herramienta matemática: Maple. De esta manera, Maple nos proporcionó una primera aproximación y nos demostró que nuestro algoritmo podía ser implementado mediante un lenguaje de alto nivel.

3.2. Estructura del programa

El programa en Maple se divide en nueve secciones cuyo código se puede consultar en el apéndice A:

1. Bernstein
2. Spline
3. Nodos intermedios
4. Subdivisión de superficies
5. Prueba de la corrección de la subdivisión
6. Algoritmo de intersección por división
7. Cálculo de la normal
8. Representación
9. Comparación de los métodos del cálculo de la normal

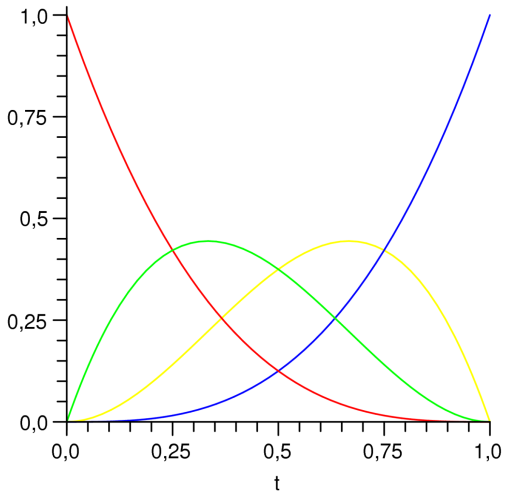


Figura 3.1: Polinomios de Bernstein (*primero*, *segundo*, *tercero* y *cuarto*) grado 3.

3.2.1. Bernstein

En esta sección hemos implementado una función recursiva para evaluar los polinomios de Bernstein de cualquier grado. Para representarlo gráficamente, hemos utilizado la herramienta *plot*. El resultado podemos observarlo en la figura 3.1.

Se puede consultar el código en el apartado A.2 del apéndice A.

3.2.2. Spline

Lo primero que aparece en esta sección es la especificación de los puntos de las mallas que vamos a utilizar como ejemplos. Estas mallas están definidas mediante sus polígonos de control, que describimos en una matriz de coordenadas. Para representar gráficamente y poder visualizar la malla, a continuación implementamos una función para dibujar el polígono. Como ejemplo, podemos observar en la figura 3.2 la siguiente malla:

$$MallaDeControl \equiv \begin{pmatrix} (0,1,1) & (1,1,1) & (2,1,1) & (3,1,1) \\ (0,2,1) & (1,2,2) & (2,2,2) & (3,2,1) \\ (0,3,1) & (1,3,2) & (2,3,3) & (3,3,1) \\ (0,4,1) & (1,4,1) & (2,4,1) & (3,4,1) \end{pmatrix}$$

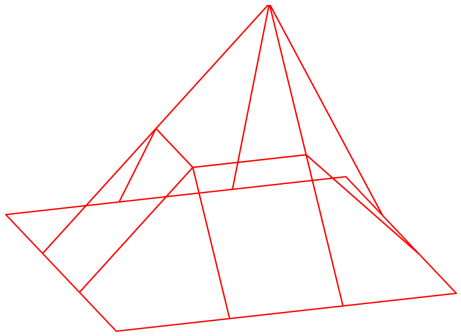


Figura 3.2: Polígono de control de la malla.

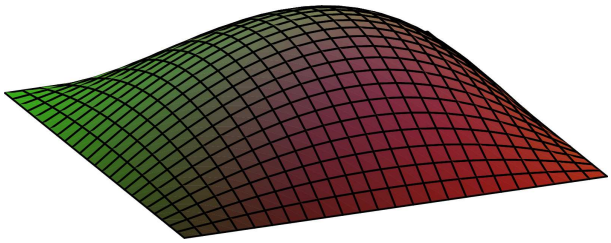


Figura 3.3: Spline correspondiente al polígono de control anterior.

A continuación, creamos una función para calcular un punto de la spline a partir de los dos parámetros u y v. Esta función nos es útil también a la hora de representar gráficamente la spline en Maple. Así, en la figura 3.3 podemos ver el resultado del dibujo de la spline correspondiente al polígono de control anterior.

Se puede consultar el código en el apartado A.3 del apéndice A.

3.2.3. Nodos intermedios

El presente apartado contiene la función encargada de calcular un nodo intermedio de una de las submallas. Un nodo intermedio se obtiene a partir de los valores del polinomio de Bernstein y puntos de la malla original. Implementamos el algoritmo de la sección 2.3.3 utilizando dos bucles anidados. Además, comprobamos que las coordenadas del nodo calculado se corresponden con las coordenadas calculadas mediante interpolación bilineal iterada del polinomio de la malla original.

Se puede consultar el código en el apartado A.4 del apéndice A.

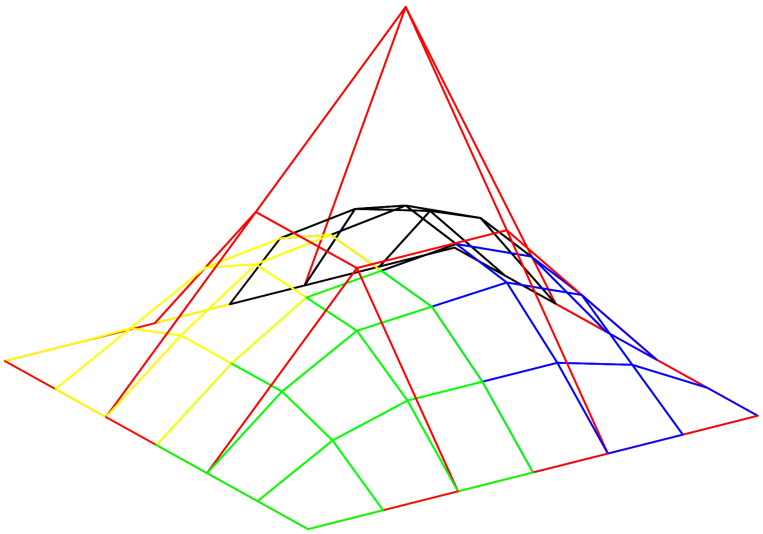


Figura 3.4: Malla de control original y sus submallas.

3.2.4. Subdivisión de superficies

La función implementada en esta sección es la función clave del algoritmo: calcula las mallas de control correspondientes a las subsuperficies de la spline original. Para obtener los nodos de cada submalla, esta función realiza una serie de llamadas a la ya mentada *calculaNodoIntermedio*. Podemos ver en las figuras 3.4 y 3.5 la representación gráfica de un polígono de control y sus submallas, así como de las subsuperficies.

Se puede consultar el código en el apartado A.5 del apéndice A.

3.2.5. Prueba de la corrección de la subdivisión

La prueba de la corrección consiste en la comprobación de que los polinomios de control de las submallas se corresponden con el polinomio de control de la malla original, salvo cambio de variable. Para ello, utilizamos dos funciones. La primera de ellas se encarga de averiguar el polinomio en u y v asociado a una spline. De esta manera, la segunda función utiliza a la anterior para la malla original y cada una de las submallas y a continuación compara los resultados aplicando un cambio de variable distinto en cada caso.

Este apartado es importante porque nos confirma, si bien no demuestra, que la subdivisión es correcta, pues permite comprobar que las submallas calculadas generan splines que son realmente trozos de la spline original.

Se puede consultar el código en el apartado A.6 del apéndice A.

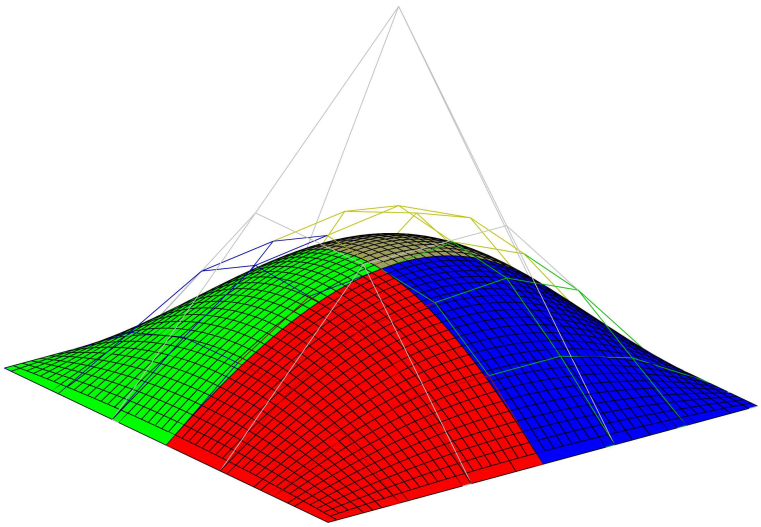


Figura 3.5: Malla de control original, las submallas y las superficies que éstas generan.

3.2.6. Algoritmo de intersección por división

El algoritmo aquí implementado se encarga, como vimos en el capítulo 2, de averiguar si una recta corta con una spline. Éste hace uso de diferentes funciones. La primera se encarga de hallar la caja mínima que contiene una spline. A continuación, definimos las funciones necesarias para trabajar con rectas, incluyendo el corte de una recta con una caja. Una parte importante de las funciones de las rectas es la creación del haz de rectas, que simula el conjunto de rectas que van desde el observador a cada uno de los puntos de la pantalla.

La función del algoritmo de intersección fue implementada de dos modos diferentes: uno iterativo y otro recursivo. La función iterativa debía implementar su propia pila para ir almacenando las submallas y las cajas, mientras que la función recursiva no necesita una estructura de datos aparte porque utiliza la propia pila de ejecución. Por tanto, la versión utilizada es la recursiva, ya que posteriormente no habría que implementar una estructura de datos.

El algoritmo de intersección comprueba si una malla corta con la recta dada, en ese caso divide la malla en sus cuatro submallas y repite el método recursivamente hasta la precisión deseada. La función devuelve el conjunto de puntos de corte de la recta con la spline.

A continuación, debemos filtrar el conjunto de puntos de corte, para ello implementamos el *Buffer Z* en la función *filtraPuntos*. Este algoritmo elige

de entre todos los puntos de corte el que tiene menor coordenada z, puesto que desde la posición del observador éste será el único punto que él pueda observar.

En la figura 3.6 podemos observar: el corte de la recta con la spline, su malla y la caja que la contiene. A continuación, vemos el cálculo recursivo de las submallas del algoritmo de intersección según el corte de la recta. Y, por último, la misma ejecución del algoritmo mostrando las cajas contenedoras de las mallas.

Se puede consultar el código en los apartados A.7 y A.11 del apéndice A.

3.2.7. Cálculo de la normal

Tras calcular un punto de corte, hemos llegado a una submalla tan pequeña como la precisión indique. Es de ésta submalla final de la que necesitamos calcular la normal.

Primeramente, el cálculo de la normal requirió un conjunto de funciones para el manejo de vectores, que se pueden ver en *Operaciones con vectores* en el apartado A.8 del apéndice A.

A continuación, implementamos tres métodos para el cálculo de la normal:

- normal al punto evaluado en $(u, v) = (0.5, 0.5)$ de la spline calculada mediante las derivadas de Bézier.
- normal a una superficie dada por tres puntos.
- normal calculada como combinación —suma— de las normales en dos puntos.

En la figura 3.7 podemos ver el método de cálculo aproximado como combinación de las normales en dos puntos y el método absoluto mediante las derivadas de Bézier.

Se puede consultar el código en los apartados A.9 y A.10 del apéndice A.

3.2.8. Representación

Una vez resuelto el cálculo de las normales ya podemos calcular el color que percibiría un observador bajo una luz determinada. Lo primero, es calcular los puntos de corte para un haz de rectas y, a continuación, filtrar los puntos de corte para cada recta del haz, quedándonos con el más cercano al observador.

El siguiente paso es el cálculo de la percepción que tendría el observador de cada corte. Para ello, utilizando la normal a la superficie, el punto de corte

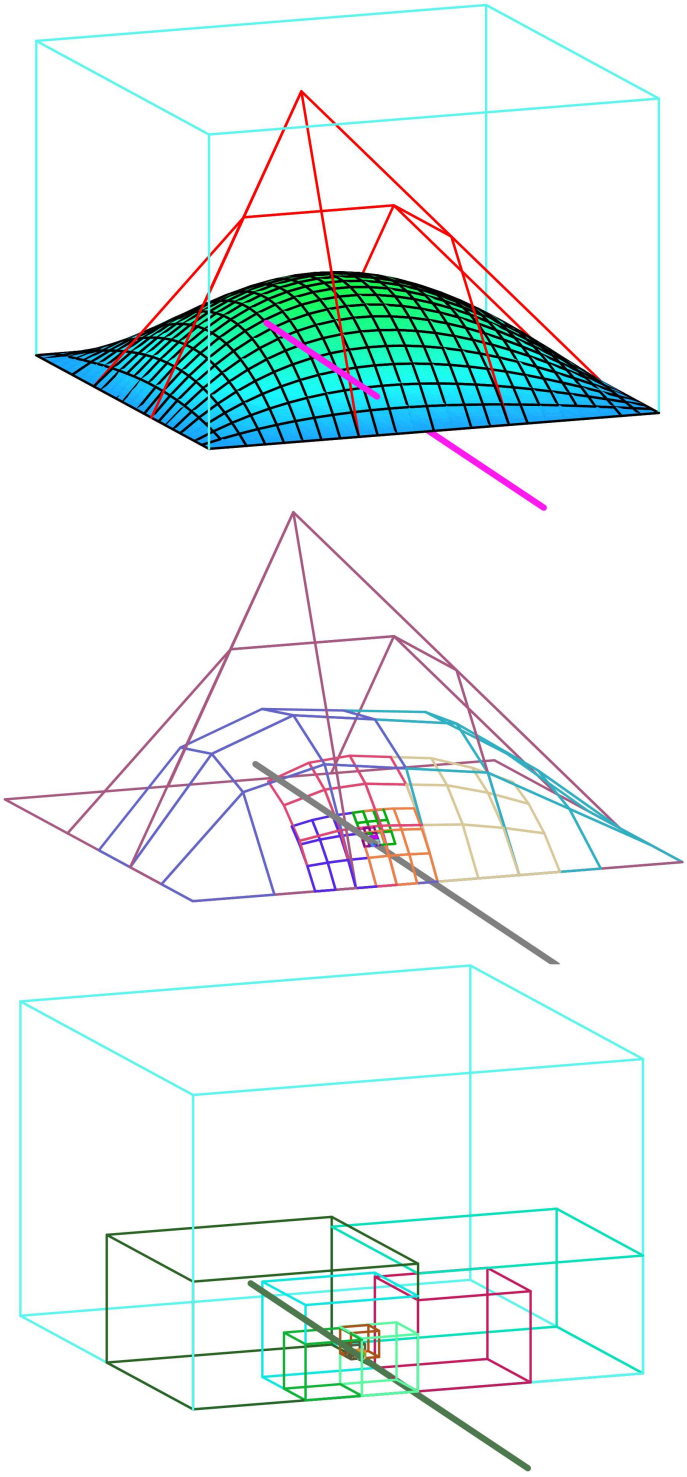


Figura 3.6: Corte de la recta con: la spline, las mallas y las cajas contenedoras calculadas con el algoritmo de intersección.

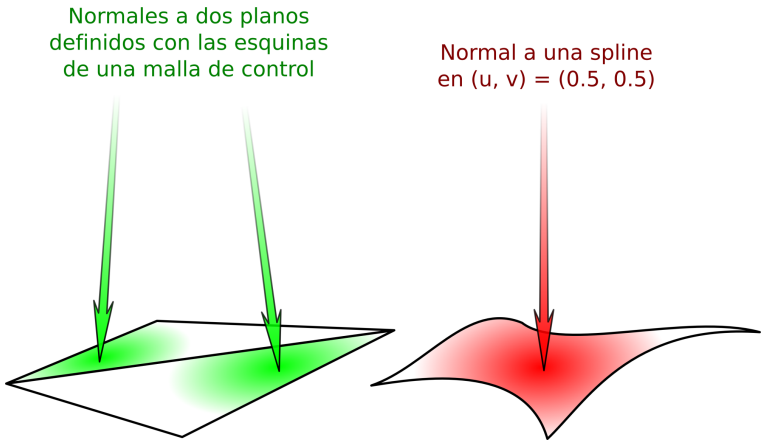


Figura 3.7: Normales calculadas de forma aproximativa y de forma absoluta.

y el foco de la luz, calculamos la cantidad de luz reflejada de cada fuente presente que llegaría hasta el observador. Finalmente, el color observado es la suma de todas las luces reflejadas. La imagen de la figura 3.8 ilustra la intersección del haz de rectas con una spline.

Se puede consultar el código en los apartados A.12 y A.13 del apéndice A.

3.2.9. Comparación de los métodos del cálculo de la normal

Esta sección fue pensada para comparar los dos métodos del cálculo de la normal. La normal evaluada mediante derivadas de Bézier es la más exacta, pero también la más costosa. Por otro lado, la normal calculada de forma aproximada como combinación de normales es más sencilla pero inexacta. Sin embargo, ya que el parche final es muy pequeño, se puede considerar plano y las normales a dicho plano serán muy parecidas a la normal exacta.

Para comprobar nuestra suposición, realizamos los cálculos mediante las dos opciones y realizamos un estudio estadístico[1] (media, varianza y desviación típica, que podemos consultar en el apartado A.15). Así, nos cercioramos de que ambos resultados se aproximaban suficientemente. Por tanto, este resultado nos permitió tomar la decisión de utilizar el cálculo de normales aproximado y así simplificar los cálculos. Los resultados observados fueron los siguientes:

[1]Los datos que analizamos estadísticamente fueron, para cada submalla que tomamos como punto de corte, el ángulo formado entre la normal a dicha malla en su centro y la normal calculada mediante la técnica aproximativa. Los resultados obtenidos fueron en radianes.

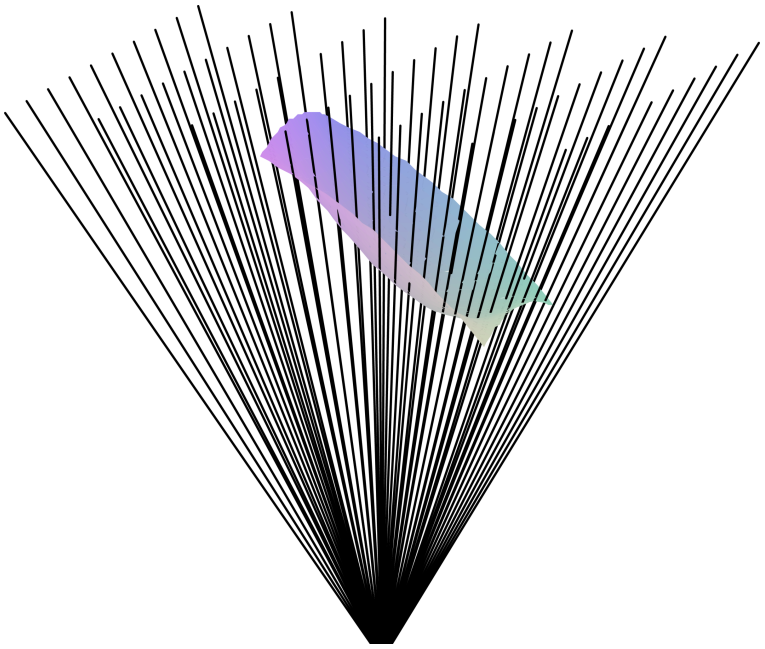


Figura 3.8: La imagen que percibe el observador se calcula a partir de los puntos de corte del haz de rectas de visión con la spline.

Tamaño muestral (normales)	**Media (diferencia)**	**Varianza**	**Desviación típica**
12	$3{,}469 \cdot 10^{-4}$	$6{,}906 \cdot 10^{-8}$	$2{,}628 \cdot 10^{-4}$
48	$2{,}259 \cdot 10^{-4}$	$4{,}299 \cdot 10^{-8}$	$2{,}073 \cdot 10^{-4}$
192	$3{,}616 \cdot 10^{-4}$	$3{,}386 \cdot 10^{-7}$	$5{,}812 \cdot 10^{-4}$
300	$4{,}111 \cdot 10^{-4}$	$3{,}647 \cdot 10^{-7}$	$6{,}039 \cdot 10^{-4}$
432	$5{,}077 \cdot 10^{-4}$	$1{,}098 \cdot 10^{-6}$	$1{,}048 \cdot 10^{-3}$
588	$4{,}528 \cdot 10^{-4}$	$5{,}22 \cdot 10^{-7}$	$7{,}225 \cdot 10^{-4}$
768	$4{,}123 \cdot 10^{-4}$	$5{,}045 \cdot 10^{-7}$	$7{,}103 \cdot 10^{-4}$
972	$4{,}781 \cdot 10^{-4}$	$1{,}473 \cdot 10^{-6}$	$1{,}213 \cdot 10^{-3}$
1200	$4{,}069 \cdot 10^{-4}$	$4{,}198 \cdot 10^{-7}$	$6{,}479 \cdot 10^{-4}$
1452	$4{,}360 \cdot 10^{-4}$	$7{,}059 \cdot 10^{-7}$	$8{,}401 \cdot 10^{-4}$
1728	$4{,}398 \cdot 10^{-4}$	$6{,}061 \cdot 10^{-7}$	$7{,}785 \cdot 10^{-4}$

Como se aprecia en la tabla, aunque hay algunas fluctuaciones, la desviación de las normales aproximadas respecto a las normales *auténticas* es mínima, y lo más probable es que no repercuta en la cantidad reflejada de luz en una magnitud medible. En la figura 3.9 podemos observar la normal calculada para una spline ejemplo.

Se puede consultar el código en los apartados A.14 y A.15 del apéndice A.

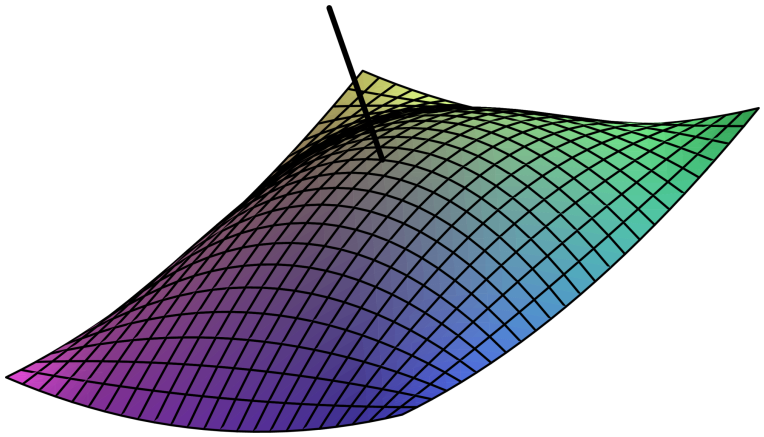


Figura 3.9: Spline y su normal.

Parte II

Desarrollo software

Capítulo 4

Implementación en C++

4.1. Introducción

Tras realizar el desarrollo del algoritmo utilizando Maple, decidimos implementar en un lenguaje de más bajo nivel todo lo que habíamos aprendido, con el fin de crear un programa que fuese capaz de mostrar una o varias splines con una o más fuentes de luz. Elegimos C++ porque nos proporcionaba la posibilidad de trabajar tanto a alto nivel —con vistas a desarrollar lo más rápido posible— como a bajo nivel, de modo que pudiésemos adaptarlo con facilidad a un posible hardware.

Realizamos el desarrollo en Windows y Linux, utilizando el entorno de desarrollo *KDevelop* en Linux y *Microsoft Visual Studio* en Windows. Para poder representar en pantalla, decidimos utilizar la biblioteca *Simple Direct-Media Layer*[1], que al soportar Windows y Linux nos permitía que nuestro código se pudiera ejecutar en ambas plataformas sin tener que modificarlo en absoluto.

La implementación que realizamos básicamente fue una "traducción" del código Maple a C++ en la mayoría de las funciones, aunque lo estructuramos de forma que se pudiera representar directamente una spline, en lugar de ser una serie de operaciones aisladas como en Maple.

4.2. Implementación

A la hora de implementar procedimos incrementalmente. Primero, programamos las operaciones de subdivisión de mallas, de generación del haz de rectas y del cálculo para determinar si una recta corta a una caja. Una

[1] `http://www.libsdl.org`

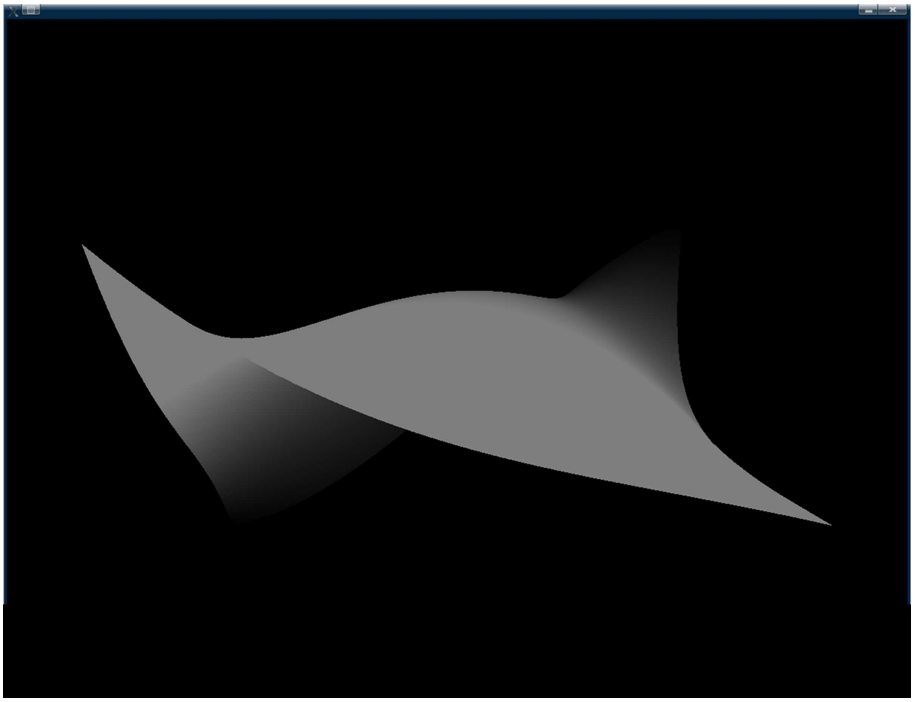


Figura 4.1: Representación de una spline en función de la distancia de cada punto al observador.

vez tuvimos esto, fuimos capaces de calcular todos los puntos de corte de un haz de rectas con una spline. Representamos entonces una spline, asignando colores a los pixels en función de la distancia entre el observador y los puntos de corte —blanco para los más cercanos, negro para los más lejanos–. El resultado se puede ver en la figura 4.1.

Una vez conseguimos representar imágenes de esta forma, procedimos a dibujar imágenes en color. Para ello, programamos el cálculo de la luz reflejada por la superficie. Necesitamos entonces ser capaces de calcular, en primer lugar, la normal a una spline. Como vimos en la sección 3.2.9, bastaba con aproximar la normal a la spline por la normal a un plano formado por tres vértices de la malla de control. Después, se programó el cálculo del porcentaje de luz difusa reflejada dados un observador, un foco, un punto de una superficie y una normal, con lo que pudimos representar finalmente una spline como la que se ve en la figura 4.2.

En último término, cuando estuvo listo todo lo anterior, programamos el cálculo de la luz especular, con lo que finalmente las splines se representaron como vemos en la figura 4.3.

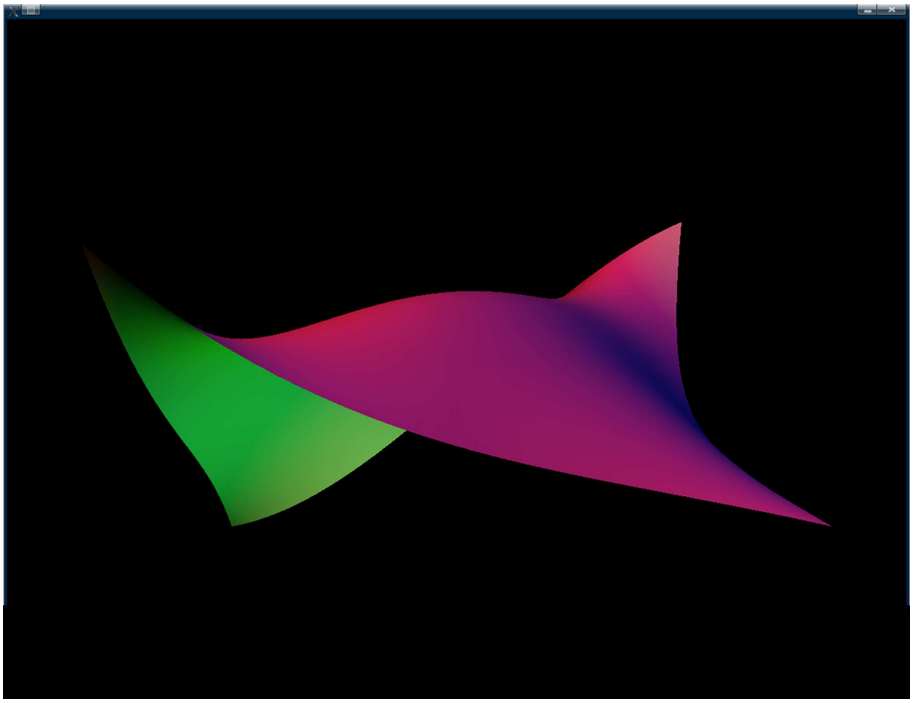


Figura 4.2: Representación de una spline con iluminación difusa.

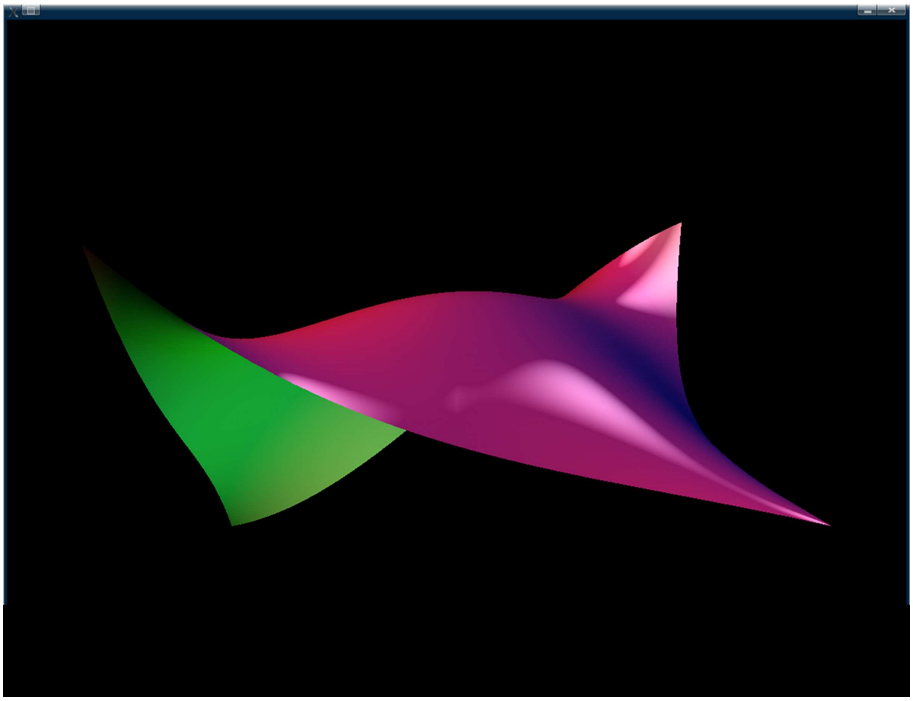


Figura 4.3: Representación de una spline con iluminación difusa y especular.

4.3. Estructuras de datos

La implementación concreta de las clases que utilizamos, y la documentación —que escribimos siguiendo el estándar de Doxygen[2]— se puede ver en el listado de código del apéndice B. Resumimos aquí el propósito de cada clase:

- *Bernstein*
  - Descripción:
    - Representación de los polinomios de Bernstein evaluados en $t = 0{,}5$. Precalcula y almacena diversos valores que se utilizan en el cálculo de las submallas, de modo que no haya que calcular los valores cada vez que se necesiten.
  - Métodos:

    calculaBernstein Calcula el valor de Bernstein para un grado y un subíndice datos en $t = 0{,}5$.

- *Caja*
  - Descripción:
    - Representación de la caja que contiene una spline. Se representa por dos puntos situados en esquinas opuestas, así como un tercer punto precalculado indicando su centro. Dicho punto se almacena para no tener que calcularlo numerosas veces.

- *Corte*
  - Descripción:
    - Representación del corte entre una recta y una spline. Cada corte se representa por un punto de corte y un vector normal a la superficie en dicho punto.

- *Imagen*
  - Descripción:
    - Representación de una escena que contiene varias mallas de control y varias luces.
  - Métodos:

[2] `http://www.stack.nl/~dimitri/doxygen/`

calcularLuces Dado un punto de corte, este método calcula el color que el observador debería percibir.

computarPixel Dadas unas coordenadas de pantalla, calcula el corte entre la recta del observador y la spline.

computarImagen Computa la imagen entera.

- *Luz*
  - Descripción:
    - Representación de una luz por su posición, valores de intensidad en colores rojo, verde y azul, así como un parámetro adicional para indicar la intensidad total de la luz.
  - Métodos:

    calculoPorc Dado un vector normal y un punto, calcula el porcentaje de luz reflejada que llega hasta el observador.

    calculoLuzR, calculoLuzG, calculoLuzB Calcula el color que debe percibir un observador, medido en cantidad de luz roja, verde y azul, respectivamente.

- *MallaControl*
  - Descripción:
    - Representación de una malla de control. La malla se representa por los 16 nodos de control.
  - Métodos:

    dividir Divide la malla en sus 4 submallas.

    calculaCorte Calcula el corte entre una recta dada y la malla de control.

    nodoIntermedio Implementa el cálculo de un nodo intermedio, siguiendo la notación de índices y la definición vistas en la sección 2.3.3.

- *Punto*
  - Descripción:
    - Representación de un punto en el espacio. Se representa por sus tres coordenadas.

- *Recta*

- Descripción:
  - Representación de una recta en forma paramétrica. Los parámetros a, b, c, d, e, f de la clase denotan a la recta:

$$[a \cdot t + b, c \cdot t + d, e \cdot t + f]$$

- Métodos:

  corta Comprueba si la recta corta con una caja dada.

  creaHazRectas Método estático que crea el haz de rectas que salen del observador y pasan por cada uno de los puntos de la pantalla.

- *Vec*
  - Descripción:
    - Representación de un vector.
- *Pixel*
  - Descripción:
    - Estructura auxiliar para el almacenamiento de un pixel, utilizando *unsigned char* para almacenar los valores rojo, verde y azul del mismo.

4.4. Algoritmo

La forma en que opera el programa, en líneas generales, es la siguiente:

1. Para cada columna de la imagen, y para cada fila, se toma la recta que parte del observador y pasa por dicho punto de la imagen.
2. Se comprueba si la recta corta con la caja contenedora de la spline.
   a) Si no corta, el pixel observado es negro.
   b) Si corta, se calcula el tamaño de la caja.
      1) Si su tamaño aparente es mayor que un pixel, se divide la malla en 4 y se calcula recursivamente el corte con cada una de las cuatro cajas. Se selecciona, de todos ellos, el punto de corte más cercano al observador.
      2) Si el tamaño aparente es menor o igual que un pixel, podemos aproximar el punto de corte por el centro de la caja. A continuación, calculamos la normal a la superficie spline y, utilizando la información obtenida, el color del pixel.

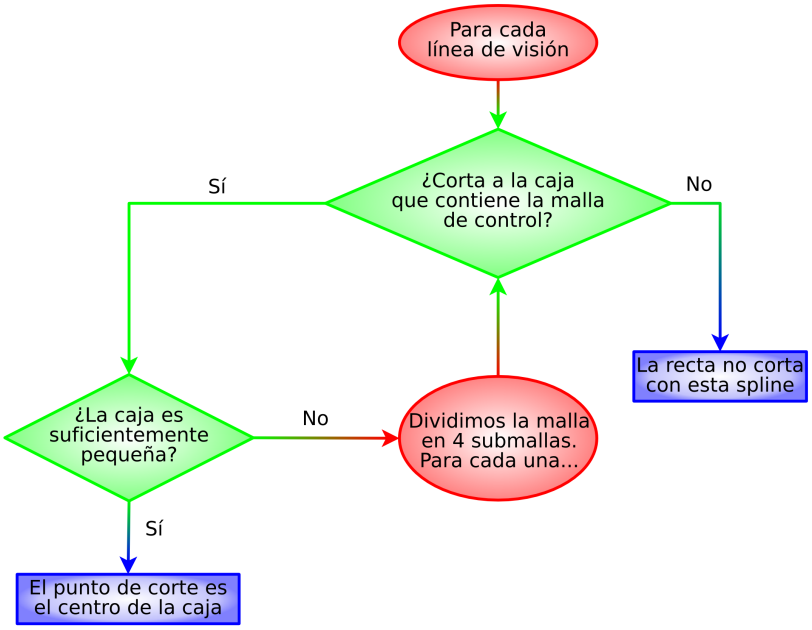


Figura 4.4: Diagrama de flujo del algoritmo.

4.4.1. Optimizaciones

Cuando por primera vez ejecutamos el algoritmo, lo primero que nos sorprendió fue que la ejecución resultó muchísimo más lenta de lo esperado. Esto se debía, principalmente, a que un gran número de operaciones se realizaban muchas veces. Por ejemplo, en el caso de que la caja de una spline ocupe toda la pantalla, todas las rectas cortarán con ella y, por lo tanto, habrá que dividir esa spline una vez por cada recta. La situación era la que se muestra en la figura 4.5.

Debido a este problema, decidimos rediseñar el algoritmo utilizando la técnica de la programación dinámica: cada malla tiene 4 punteros que inicialmente son nulos pero que, cuando se divide la malla, apuntan a sus 4 submallas. Almacenamos en memoria un árbol de mallas, de modo que cuando se trata de calcular un punto de corte, cada subdivisión se realiza una sola vez. Si al comprobar el corte ya se realizó la subdivisión anteriormente, basta con comprobar con las mallas hijas sin necesidad de dividir. Pasamos de una situación como la de la figura 4.5 a una situación del estilo de la figura 4.6.

Sin embargo, este nuevo enfoque tenía otro problema: el elevado espacio en memoria requerido. Por ejemplo, una imagen con una resolución de 800×600 pixels necesitaba un espacio en memoria, sólo para almacenar una malla y sus submallas, de varios cientos de Mbytes. Esto hacía que la técnica

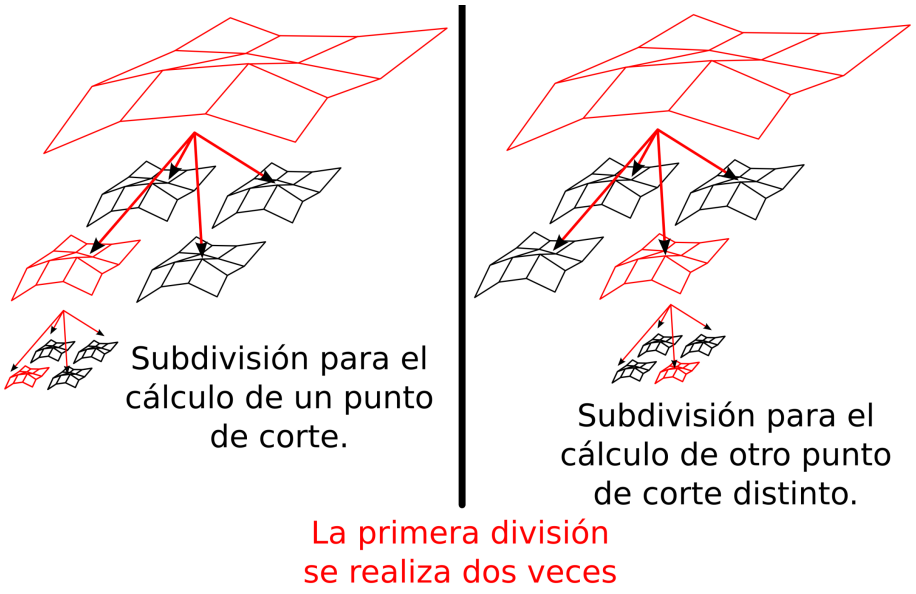


Figura 4.5: Esquema del cálculo original.

fuese totalmente infactible en la práctica, pues el hardware que finalmente utilizamos tenía una capacidad mucho menor.

Para que nos hagamos una idea del tamaño que podía llegar a ocupar en memoria nuestra estructura, supongamos una spline que, al representarla, ocupe toda la superficie de una pantalla de $640 \times 480 = 307200$ pixels. Si, para calcular cada corte —tantos como pixels—, tenemos que dividir la malla original hasta que las cajas que contienen las mallas sean tan pequeñas como los pixels, esto nos dará un total de, sólo en las hojas de la estructura del árbol, otras 307200 mallas. Un punto en el espacio se representa por sus coordenadas (x, y, z) utilizando 3 números de 32 bits; un total de 96 bits por punto. Si cada malla se compone por 16 puntos, las mallas de las hojas del árbol de cálculo ocuparán en memoria un total de

$$307200 \cdot 16 \cdot 96 = 471859200 \quad bits$$

o lo que es lo mismo, más de 56Mbytes, y todo ello sin contar los nodos intermedios del árbol, los punteros entre nodos, las cajas contenedoras, etc.

Por lo tanto, la primera optimización que realizamos fue eliminar los nodos de las hojas; una vez que la malla cabe en una caja de un tamaño suficientemente pequeño, ya no es necesario volverla a dividir. Dado que la división se realiza a partir de sus nodos, si ya no es necesario realizarla se pueden eliminar los puntos de control. Tras realizar esta optimización, el consumo de memoria se redujo considerablemente, pero no lo suficiente:

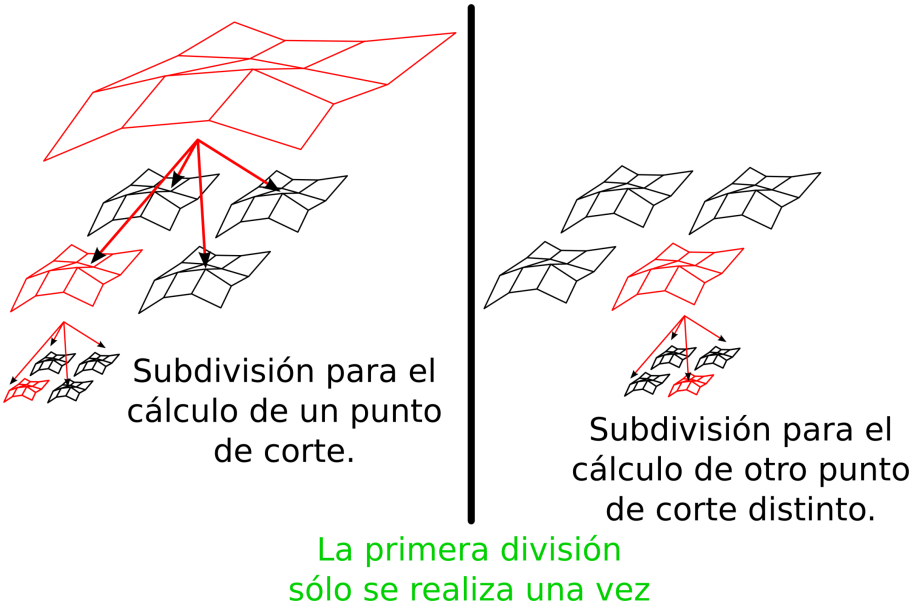


Figura 4.6: Esquema del cálculo que se realiza tras aplicar la técnica de programación dinámica.

tuvimos que optimizar el consumo de los nodos intermedios del árbol de mallas.

Supongamos de nuevo una spline que ocupa exactamente una superficie en pantalla de $640 \cdot 480 = 307200$pixels. Al dividir la superficie original en 4 partes, tendríamos que dividir la superficie original $\log_2 640 = 9{,}32 \simeq 10$ veces[3]. Por lo tanto, el árbol de cálculo tendrá, aproximadamente, 9 niveles de nodos intermedios. Calculando, vemos que ocuparía un total de

$$16 \cdot 96 \cdot \sum_{i=0}^{9} 4^i = 536870400 \quad bits$$

que, en bytes, suponen casi 512Mbytes. Todo ello sin contar las cajas, los punteros entre nodos, etc.

La solución vino de nuevo de la programación dinámica: si ya sólo dividimos las mallas una vez como mucho, no tenemos por qué almacenar los vértices de control de una malla que ya hemos dividido, sólo su caja contenedora. Así pues, el algoritmo se vio modificado de la siguiente forma:

[3]Aunque la superficie se divida en 4 superficies, el ancho y el alto sólo se dividen por 2 cada uno. Por lo tanto, la altura del árbol se calcula con $\log_2$ y no con $\log_4$. Además, sólo tenemos en cuenta los 640 pixels porque son más que los 480.

1. Al calcular el corte entre una recta y una malla, primero comprobamos si la recta corta a la caja contenedora.

2. Si efectivamente la corta, entonces comprobamos el tamaño de la misma.

   a) Si es suficientemente pequeña, nos quedamos con el centro de la caja y eliminamos los vértices de la malla de control.

   b) Si no lo es, dividimos la malla si no había sido dividida antes, y eliminamos los vértices de la malla de control.

Aunque pueda parecer, por los cálculos anteriores, que el espacio en memoria no se reduce tanto como se desearía, en la práctica resulta muy satisfactorio, pues las mallas se dividen en la práctica mucho más que en los ejemplos teóricos expuestos. La única memoria que queda utilizada inútilmente ahora es la que ocupan los nodos del árbol que contienen mallas a las que no corta ninguna recta.

4.4.2. Ajustes finales

Una vez reducido el consumo de memoria y el tiempo de cálculo, quedaba por realizar una mejora más. Hasta este momento, el tamaño mínimo de las cajas estaba determinado por un valor arbitrario que fuimos ajustando nosotros. Para ello, elegimos un cierto valor y fuimos reduciéndolo, tratando de producir los efectos deseados, como se puede ver en la figura 4.7.

El problema es que, para que la imagen se vea bien, utilizar un tamaño de caja fijo implica emplear más recursos de los estrictamente necesarios. Por ejemplo, si tenemos una situación como la de la figura 4.8, las cajas de los puntos cercanos al observador son del tamaño adecuado, mientras que las cajas de las mallas alejadas son innecesariamente pequeñas; esto hace que se calcule el punto de corte con una precisión mayor de lo que se puede apreciar finalmente en la pantalla.

Lo que hicimos fue modificar el algoritmo para que fuese adaptativo: cuanto más cercana al observador esté una caja, más pequeña debe ser. La idea es que cada caja tenga un tamaño aparente de, como mucho, un pixel. De este modo, el observador siempre verá la imagen con todo detalle, y no se realizarán cálculos para conseguir una precisión más allá de lo apreciable.

Para hacer que el algoritmo sólo divida las mallas hasta alcanzar el nivel del tamaño de un pixel, lo que hacemos es lo siguiente:

1. Suponemos que el plano de proyección es el plano $z = 1$.

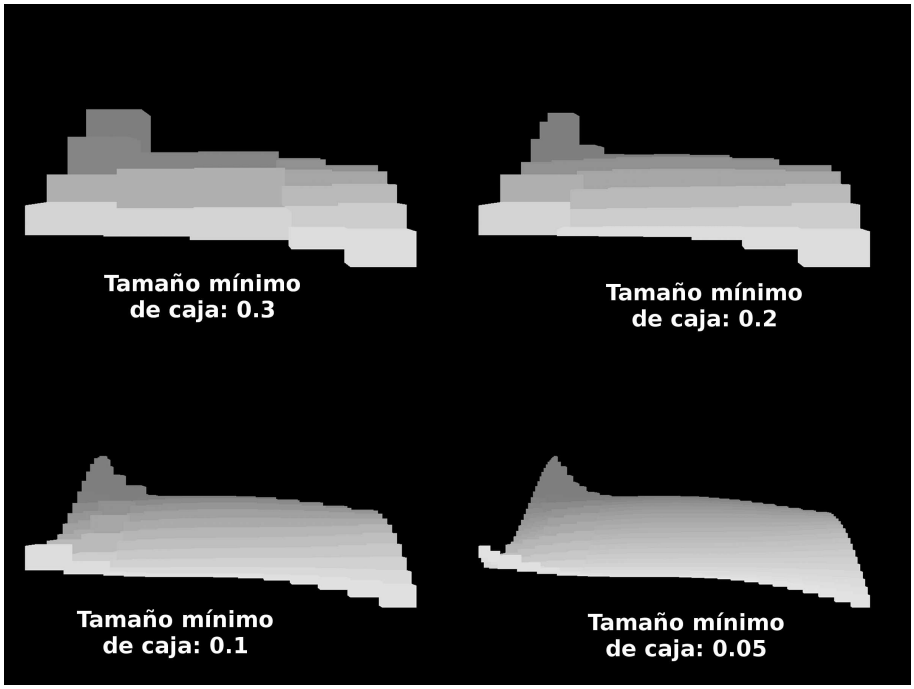


Figura 4.7: Imágenes de las pruebas para el ajuste del tamaño de la caja mínima.

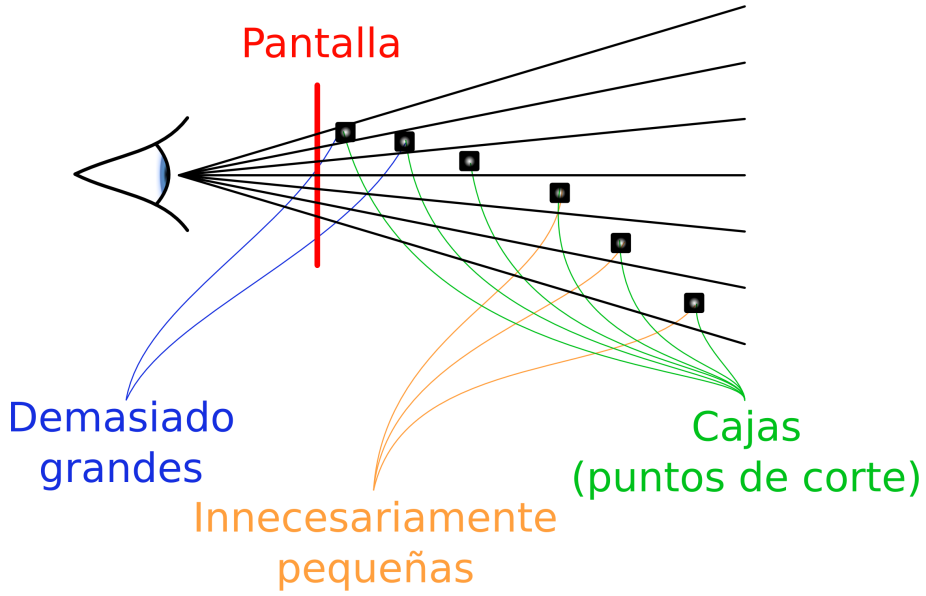


Figura 4.8: Si todas las cajas son del mismo tamaño, las que están muy lejos del observador son innecesariamente pequeñas.

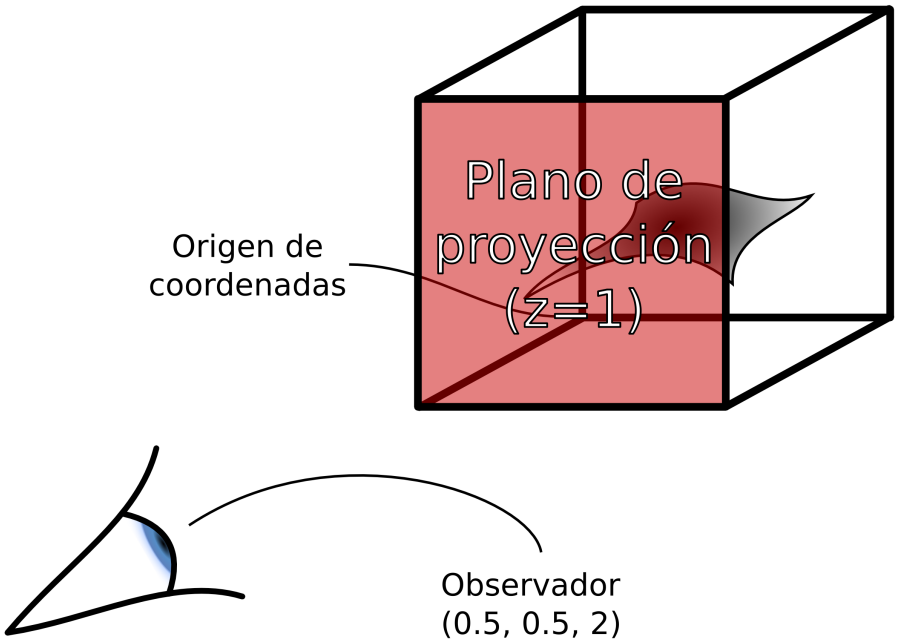


Figura 4.9: Colocación de los elementos de la escena.

2. Dividimos el ancho del plano de proyección ($z = 1$) entre el número de pixels que tenga la imagen de ancho, (640 en nuestro caso).

3. Suponemos que la anchura de una caja que se encuentre en el plano del observador ($z = 2$) será 0. La situación de la escena se puede ver en la figura 4.9.

De este modo, podemos establecer el tamaño de las cajas en función de su distancia al observador utilizando una relación lineal:

$$\text{tamaño}_{\text{máximo}} = anchura \cdot centro_z + anchura$$

donde:

- *anchura* es la anchura de cada pixel en coordenadas de la escena, esto es:
$$\frac{1}{pixels_{ancho}}$$
- $centro_z$ es la coordenada z del centro de la caja.

4.4.3. Depuración

Para asegurarnos de que la gestión de memoria era totalmente correcta, utilizamos un programa capaz de "vigilar" la utilización de memoria: Valgrind[4]. Este programa sirve para observar el uso de memoria que otro programa hace; para ello, cada vez que el programa reserva, libera, escribe y lee memoria, comprueba que esa memoria haya sido correctamente utilizada.

Gracias a Valgrind, pudimos depurar una serie de pérdidas de memoria y de lecturas de variables no inicializadas que, de otro modo, podría habernos llevado mucho más tiempo o, incluso, no ser conscientes de que nuestro programa no era correcto.

Para la depuración, utilizamos GDB[5], que nos permitió corregir el comportamiento del programa en ciertas situaciones en las que no realizaba los cálculos correctamente, debido a pequeños errores de programación.

[4] `http://valgrind.org`
[5] `http://sourceware.org/gdb/`

Parte III

Desarrollo hardware

Capítulo 5

Diseño hardware

5.1. Introducción

Una vez terminamos el programa en C++ para representar splines, tuvimos que decidir qué tipo de hardware íbamos a diseñar para realizar el cómputo. En un principio, nos planteamos hacer un hardware que, dados los puntos de las mallas a representar y las fuentes de luz, representase la imagen sin necesidad de un controlador externo. Sin embargo, debido a la envergadura de tal diseño y el poco tiempo disponible, nos decidimos por otro tipo de enfoque.

Se nos ofreció la posibilidad de utilizar un hardware que integra un procesador de propósito general y una FPGA; concretamente una *Virtex II Pro*, que integra una FPGA con un procesador *PowerPC 405*. Dicha placa contiene una serie de periféricos que nos han resultado esenciales a la hora de realizar nuestro proyecto: una salida VGA que muestra directamente los datos de un *framebuffer*, y una memoria externa DDR de 512Mbytes.

Como nuestro propósito es demostrar que este tipo de imágenes se pueden representar rápidamente, realizamos un perfil del programa para ver qué partes eran las más lentas. Tras realizar el perfil que podemos ver en el apéndice E, llegamos a la conclusión de que teníamos 2 operaciones candidatas a ser reimplementadas por hardware: la subdivisión de mallas y la comprobación del corte entre recta y caja. Viendo que el tiempo de cómputo dedicado a la subdivisión era mucho mayor, y que su paralelismo implícito era también muy grande, decidimos finalmente implementar en hardware la subdivisión de mallas.

Es importante recordar que la subdivisión de mallas es un proceso que toma 16 nodos de una malla original y devuelve 4 mallas de 16 nodos —64 en total—, donde dichos nodos resultado sólo dependen de los datos de entrada y

no de los datos de otras salidas. Por lo tanto, si dispusiéramos de suficientes unidades funcionales, podríamos calcular los 64 nodos del resultado en el tiempo necesario para generar un único nodo. El número de operaciones necesarias para realizar la tarea completa son 2 productos y una suma en la parte más interna del bucle que podemos ver en la sección A.4. De esta manera, el número total de operaciones es de

$$Total = \sum_{i=0}^{3}\sum_{j=0}^{3}\left(\sum_{k=0}^{j}\sum_{l=0}^{i}(2+1) + \sum_{k=0}^{j}\sum_{l=0}^{3-i}(2+1)+\right.$$

$$\left.+\sum_{k=0}^{3-j}\sum_{l=0}^{i}(2+1) + \sum_{k=0}^{3-j}\sum_{l=0}^{3-i}(2+1)\right) = 1200$$

instrucciones de punto flotante.

Si dispusiéramos de un hardware capaz de realizar las operaciones en paralelo, tener algunos valores precalculados y operar de forma combinacional en lugar de secuencial —esto es, punto fijo en lugar de punto flotante— podríamos llegar a realizar la división en un sólo ciclo de reloj. En contraste, la operación de test para comprobar que la recta corte a la caja contenedora realiza sólo 6 cálculos independientes entre sí, con lo que en el caso mejor podríamos dividir el tiempo de cálculo por 6, de modo que nos decidimos por la operación de división.

Tras la elección de la operación crítica, nos pusimos a trabajar en el hardware de aritmética necesario, concretamente, multiplicadores y sumadores. Para ello, necesitábamos que estas operaciones fueran en punto flotante, cuya implementación podría requerirnos mucho más tiempo del que disponíamos.

La primera solución que barajamos fue la de utilizar un único tipo de redondeo en el punto flotante y además no utilizar signo. Sin embargo, aún así y todo el hardware necesario para abarcar todas las posibilidades de productos y sumas podría resultar tremendamente grande y complejo, y no estábamos seguros de poder completarlo en el tiempo disponible. La solución nos la proporcionó el análisis teórico del algoritmo: las splines respetan la envoltura convexa de su malla de control.

Ya que una spline se define por una malla de control conocida, y su superficie está contenida en la envoltura convexa de la malla, si limitamos la malla a un cierto tamaño, la spline también queda limitada. Aún más importante, las coordenadas de todos los nodos de la malla y de todas su subdivisiones también pertenecerán al rango en que se encuentre la malla original. Por lo tanto, si elegimos una malla que quede contenida en un cubo de una unidad de lado, todas las magnitudes estarán en el intervalo $[0, 1]$. Así,

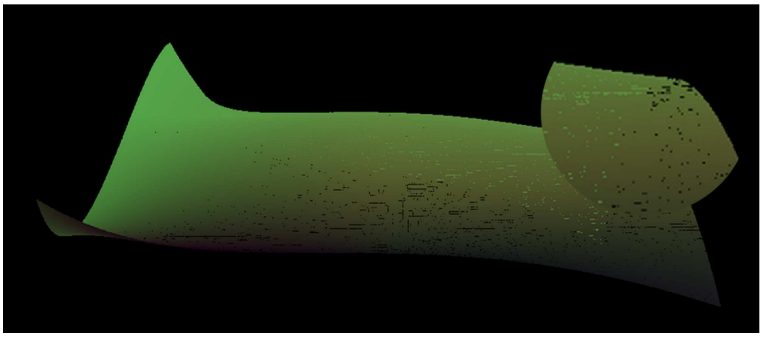


Figura 5.1: Al utilizar punto fijo de 16 bits, aparecen ciertos defectos en la imagen.

puesto que los valores se encuentran dentro de un rango acotado y además éste es pequeño, decidimos realizar los cálculos en punto fijo.

El formato numérico que utilizamos se basó en el hecho de que todos los números serían de la forma "0,...", siendo la representación del número decimal 1 en binario "0,111...1". De modo que, si utilizamos números de 32 bits, disponemos de una precisión de 32 bits decimales, la cual es más que de sobra para el trabajo que nos ocupaba. Para comprobarlo, reimplementamos la parte del cálculo de las submallas en C++ con los productos y las sumas en punto fijo, como se puede leer en la sección B.10.1. Utilizando las funciones y la representación numérica que se pueden ver en dicho apéndice, comprobamos que la imagen se calculaba perfectamente con 32 bits de precisión, si bien utilizando menos bits —en concreto 16 bits— aparecían algunos defectos, como observamos en la figura 5.1.

5.2. Multiplicador

Los sumadores en punto fijo son sumadores de enteros normales y los implementa la herramienta que utilizamos automáticamente, de modo que sólo faltaba implementar en VHDL los multiplicadores. Primero consideramos utilizar, para los multiplicadores, los que incluye la propia FPGA, aunque nos encontramos con el problema de que sólo tenían 18 bits de ancho. Sin embargo, pensamos en utilizar los multiplicadores de la placa utilizando una técnica que se explica en la página de Xilinx para encadenar los multiplicadores y que operen como si fueran multiplicadores de 36 bits, aunque más tarde descubrimos una forma más sencilla de realizar los productos.

5.2.1. Diseño del multiplicador

Diseñamos nuestro propio multiplicador con un multiplicador de Pezaris de 32 bits de entrada en el que se devolvían los 32 bits más significativos de la salida en lugar de los menos significativos. Siguiendo el esquema de multiplicación tradicional, desarrollamos una celda básica que luego replicamos mediante metaprogramación —utilizamos un programa en Java para generar el código VHDL—.

5.2.2. Optimización del multiplicador

Tras estudiar detenidamente el multiplicador realizado, aunque el resultado se obtiene con un retardo mínimo, nos dimos cuenta de que éste se podía simplificar mucho más. Cuando se calcula un nodo intermedio, como hemos visto a lo largo de los apartados anteriores, se multiplica un valor introducido por el usuario —la malla de control— por dos números conocidos —los resultantes de evaluar los polinomios de Bernstein—. Ya que dos de estos números se conocen de antemano, su producto se puede precalcular. Al observar la codificación en punto fijo de dichas constantes, nos dimos cuenta de los números eran de 4 tipos:

- El número 0 (todos los bits a 0).
- El número 1 (todos los bits a 1).
- Números con un solo bit a 1.
- Números con sólo 2 bits a 1.

De este modo, todos los números multiplicados por 0 simplemente no hay que sumarlos; los multiplicados por 1 no hay que modificarlos, y sólo quedan los otros dos tipos. Para multiplicar un número por otro en punto fijo que sólo tenga un bit a 1, basta con realizar un desplazamiento a la derecha; si los números siguen la codificación antes mencionada, si se multiplica por el número $100\ldots0$ hay que desplazar un bit a la derecha. Para multiplicar por otro que tenga 2 bits a 1, lo que hacemos es lo siguiente:

1. Sea X un número en punto fijo que se multiplica por una constante, en este caso, 0,11: $S = X \cdot 0{,}11000\ldots$
2. $S1 = X \cdot 0{,}10000\ldots \Rightarrow S1 = 0 \quad \& \quad X(31 \text{ downto } 1)$.
3. $S2 = X \cdot 0{,}01000\ldots \Rightarrow S2 = 00 \quad \& \quad X(31 \text{ downto } 2)$.

4. Se suman los dos resultados: $S = S1 + S2$.

De esta forma, pasamos a utilizar simplemente sumadores, ya que para desplazar sólo hay que conectar adecuadamente las señales.

5.3. Red de cálculo

Tras implementar los multiplicadores con sumadores, tuvimos que crear la red de sumadores que generan los resultados a partir de los 16 datos de entrada. Esta red debería implementar todas las operaciones necesarias para generar los nodos intermedios de las submallas; en C++, esto correspondía a implementar todas las operaciones que realizaba la función `nodoIntermedio`, que se puede ver en la sección A.4.

Nuestra intención es la de crear una arquitectura totalmente paralela, que realice todas las operaciones de la división de forma simultánea. El cálculo de cada nodo de las mallas resultado es independiente del resto de nodos, por lo que se puede paralelizar fácilmente. Para ello, procedimos a "desenrollar" los bucles mostrados en la sección antes citada. Básicamente, la transformación que realizamos fue la siguiente: sea un programa de la siguiente forma:

```
int total_a = 0;
int total_b = 0
for (int x = 0; x < 4; x++) {
   for (int y = 0; y < 4; y++) {
      total_a = total_a + valor[x+1][y];
      total_b = total_b + valor[y][x+1];
   }
}
```

Entonces, podemos crear un hardware en VHDL que calcule los dos valores del siguiente modo:

```
signal total_a, total_b : std_logic_vector(n - 1 downto 0);

begin

total_a <= valor10 + valor11 + ... + valor43;
total_b <= valor01 + valor11 + ... + valor34;
```

Por lo tanto, podría modificarse el bucle primero para que, directamente, generase el código VHDL:

```
String total_a = "total_a <= ";
```

```
String total_b = "total_b <= ";
for (int x = 0; x < 4; x++) {
    for (int y = 0; y < 4; i++) {
        total_a += "+valor" + String(x+1) + String(y);
        total_b += "+valor" + String(y) + String(x+1);
    }
}
print(total_a);
print(total_b);
```

Siguiendo esta técnica, procedimos a desenrollar los bucles que calculan las 4 submallas, de modo que se generase automáticamente el código necesario. Así, conseguimos implementar en VHDL un hardware relativamente denso "a prueba de fallos"; siempre que el bucle estuviese bien programado, la "traza" que genera el código VHDL generaría código hardware correcto. Por lo tanto, para depurar el diseño sólo tendríamos que depurar el código de los bucles, y no el código hardware, que es más denso y difícil de leer. Dado que, cuando llegamos a esta fase del proyecto, todo el software estaba depurado y era correcto, el diseño hardware no tuvo errores desde el principio. No obstante, realizamos simulaciones con *ModelSim*[1] para comprobar que el hardware estaba bien diseñado. Podemos ver el hardware resultante en la figura 5.2.

5.4. Protocolo de comunicación con la FPGA

Tras diseñar el hardware capaz de realizar los cálculos, procedimos a diseñar otro que fuese capaz de controlarlo y que, además, fuera capaz de comunicarse con el procesador *PowerPC* integrado en la FPGA.

El *PowerPC* y los periféricos se comunican a través de distintos tipos de buses con diversas velocidades. Estos buses tienen en común una serie de señales de control, entre las que se incluyen:

- Señal de chip–select. Sirve para identificar el periférico sobre el que el procesador está ejecutando una operación de lectura o de escritura.
- Señal de $Read/\overline{Write}$. Se utiliza para que el periférico sepa si se está solicitando una operación de lectura o de escritura.
- Bus de direcciones.
- Bus de datos.

[1] `http://www.model.com/`

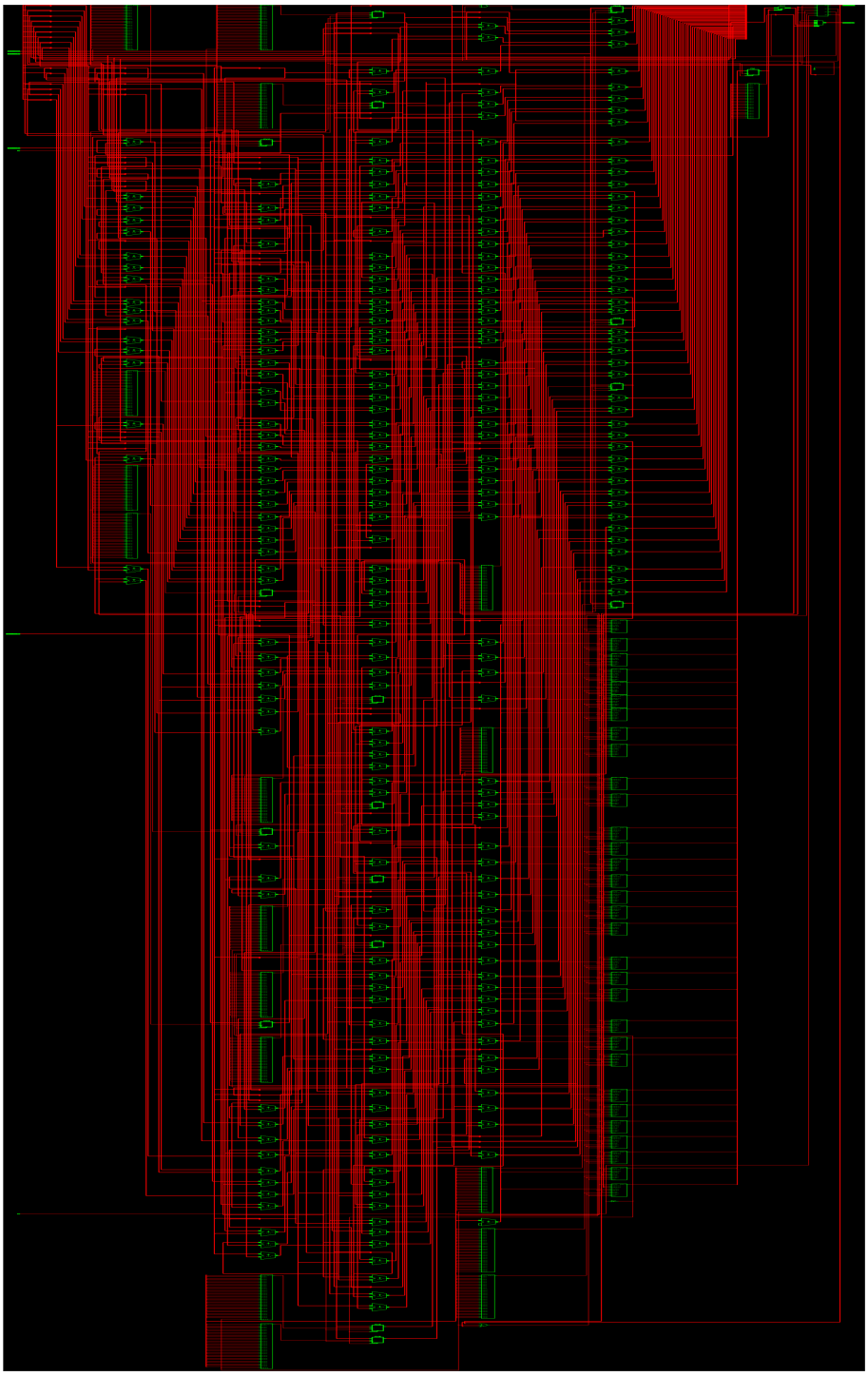


Figura 5.2: Esquemático del hardware que calcula las submallas.

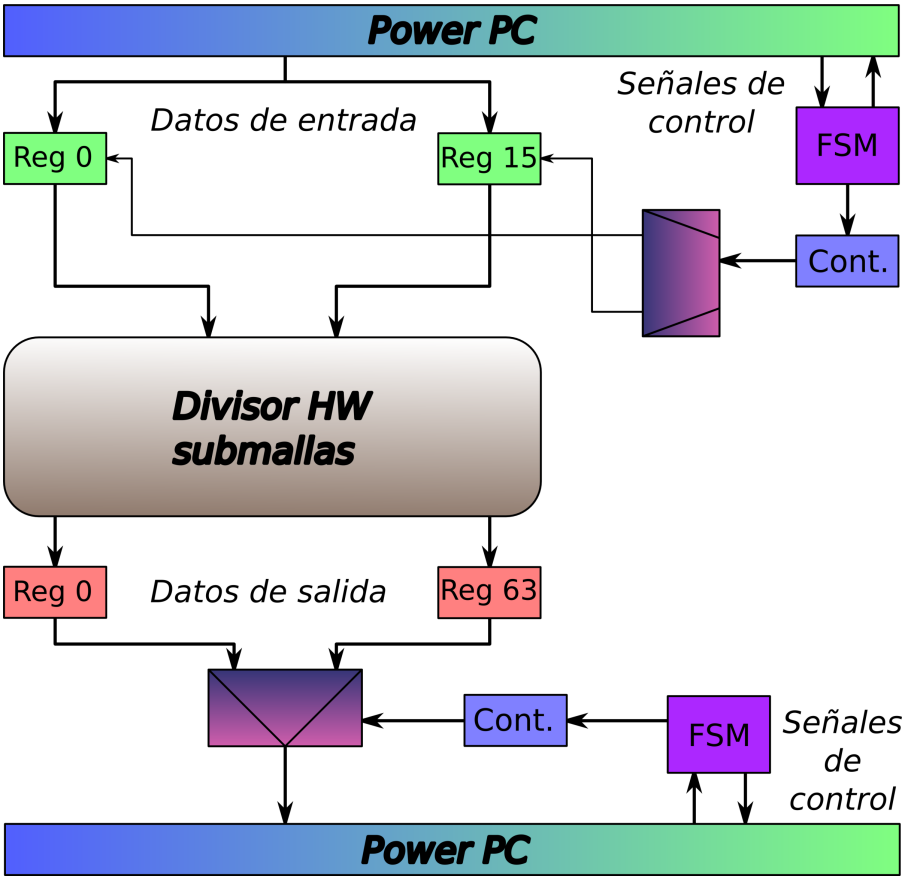


Figura 5.3: Arquitectura del protocolo de control de acceso al hardware.

- *Reset*, *Bus_Clk*, etc.

En un principio, tratamos de comprender el mecanismo a implementar para que el procesador pudiera escribir directamente los datos de entrada —la malla original— y leer los datos de salida —las submallas— a través de los buses. Sin embargo, aunque se trata de un protocolo bastante sencillo, resulta técnicamente complejo realizar el intercambio de datos mediante las señales de control vistas anteriormente. Por lo tanto, decidimos utilizar una aproximación más sencilla implementando nuestro propio controlador. El diagrama a nivel de transferencia de registros del hardware implementado se puede ver en la figura 5.3.

El controlador de nuestro hardware debería ser capaz de:

- Almacenar en registros los datos de entrada para que el hardware pueda calcular los resultados.

- Acceder ordenadamente a los resultados, de forma que el procesador pueda leer los datos calculados.

- Cuando se realice la escritura, se escribirán ininterrumpidamente todos los datos en los registros de entrada.

- Cuando se realice la lectura, también se leerán de forma ininterrumpida todos los datos.

- Cada ciclo de escritura irá siempre seguido de un ciclo de lectura.

Teniendo en cuenta todo lo anterior, implementamos el hardware de la figura 5.3, que funciona del siguiente modo: el *PowerPC* indica a la máquina de estados que quiere comenzar la escritura de datos. La máquina de estados inicializa el contador e indica al *PowerPC* que está lista. El *PowerPC* recibe la señal y manda el primer dato. El dato se almacena en el primer registro y la máquina de estados indica al *PowerPC* que se ha almacenado correctamente. Esto se repite 16 veces, almacenando los 16 datos de entrada del algoritmo de división en el registro que indique la señal que proviene del decodificador controlado por la máquina de estados y el contador. Una vez el *PowerPC* termina de escribir todos los datos de entrada, comienza a leer los resultados. Aunque los resultados tardan en estar listos entre 3 y 4 ciclos de reloj, como el *PowerPC* tiene que realizar una serie de inicializaciones de bucles para comenzar la lectura, no es necesario utilizar ningún mecanismo que fuerce la espera. En este momento se comienzan los ciclos de lectura. La lectura de los 64 datos de salida se realiza de forma parecida a la escritura: el *PowerPC* indica que quiere leer y la máquina de estados se encarga de poner el dato a su disposición a través de un multiplexor controlado por ella y por el contador. Esta vez, el contador es módulo 64 y también es controlado por la máquina de estados. Tras las 64 lecturas, la máquina de estados pasa a un estado de reposo preparada para una nueva comunicación.

El diagrama de flujo de la máquina de estados (FSM) es el que vemos en la figura 5.4. Su funcionamiento es el siguiente:

1. Inicialmente, el controlador espera que comience un ciclo de escritura. Este ciclo se repite 16 veces.

   a) El procesador carga en el bus de datos el dato a almacenar en el hardware.

   b) Cuando el procesador debe escribir, se lo indica al hardware afirmando una señal **Escribir**.

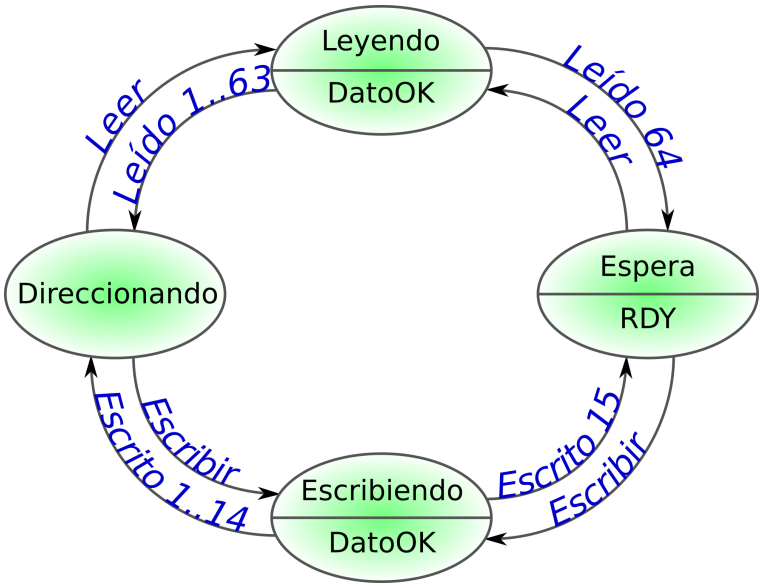


Figura 5.4: Diagrama de estados del protocolo de acceso al bus.

c) Tras recibir la señal, el controlador almacena el dato y confirma la escritura afirmando una señal **Data OK**.

d) El procesador recibe la confirmación de escritura (**Data OK**), por lo que niega la señal **Escribir** y se da por enterado, afirmando la señal **Escrito**. Procede además a cargar el siguiente dato en el bus de datos.

e) El hardware reconoce que el procesador conoce la situación, al ver que está afirmada la señal **Escrito**. Pasa a un estado de espera.

2. Tras completar las 16 ejecuciones del ciclo de escritura, el controlador pasa a un estado de reposo en el que espera que se produzca un ciclo de lectura. Dicho ciclo se compone de 64 iteraciones del protocolo siguiente:

a) El procesador indica al controlador que desea obtener un dato resultado. Para ello, afirma la señal **Leer**.

b) El controlador coloca en el bus de datos el dato calculado. Indica al procesador que el dato está listo afirmando una señal **Data OK**.

c) Tras recibir la señal **Data OK**, el procesador lee el dato presente en el bus. Notifica al controlador que ha leído y que puede preparar otro dato afirmando la señal **Leído**. Pasa a un estado de espera.

3. Tras realizar 64 iteraciones[2], el controlador vuelve al estado inicial, preparado para realizar otro ciclo de lectura–escritura.

El controlador, además de por la máquina de estados descrita, se compone por los siguientes elementos:

- 16 registros de entrada para almacenar los datos de la malla original.
- Un contador de control módulo 16 / 64, que se utiliza para contabilizar las 16 escrituras y 64 lecturas.
- Un descodificador que selecciona el registro en el que se guardan los datos leídos del bus, en función del valor que indique el contador.
- Un multiplexor que selecciona qué salida de la red de cálculo se vuelca en el bus de datos —operación de lectura—, en función del valor indicado en el contador de control.

Finalmente, decidimos implementar un controlador que utilizase las señales de control antes indicadas, y conectarlo a los pines de GPIO del procesador.

5.4.1. GPIO

GPIO son las siglas de *General Purpose Input / Output*. Se trata de un tipo de dispositivo que se utiliza, sobre todo, en sistemas empotrados, aunque también se incluye en muchos procesadores de propósito general. Un GPIO proporciona un conjunto de puertos de entrada / salida, configurables por el usuario. Esto se consigue de la siguiente manera:

- Los pines de salida se conectan a un buffer tri-estado y a un registro, el registro de entrada.
- Cuando se utiliza como entrada, el buffer tri-estado se configura en alta impedancia, de forma que el registro de entrada permite leer los valores que otro dispositivo cargue en el GPIO.
- En el caso de la salida, el buffer tri-estado fuerza en los pines los valores indicados por el usuario.

De este modo, cada pin del GPIO se puede utilizar como entrada o salida de forma independiente. Además, normalmente el GPIO incluye un registro y una conexión con una línea de interrupción, de modo que genere interrupciones cuando el dispositivo al que se conecte el GPIO fuerce ciertos valores.

[2]Los datos calculados son los 16 nodos de las 4 submallas, esto es, $16 \cdot 4 = 64$ valores.

Capítulo 6

Implementación en VHDL

6.1. Introducción

Como explicamos en el capítulo dedicado al diseño hardware, la mayor parte de éste fue implementado a partir del desenrollado de bucles mediante una traza. No obstante, no se trata de realizar una simple traza, sino que hay que generar señales intermedias, almacenar sus nombres, y realizar otra serie de cálculos auxiliares. Además, en las primeras fases también implementamos un multiplicador de Pezaris, totalmente funcional, utilizando esta misma técnica, aunque finalmente no fue necesario utilizarlo.

6.2. Creación de la red de cálculo en Java

Para la red de cálculo, como ya se indicó, procedimos a crear una traza en el programa creado en C++. Inicialmente, la traza generaba código también en C++, para así poder sustituirlo fácilmente por los bucles y comprobar su corrección. A sabiendas de que esta técnica era factible, la utilizamos para generar el código hardware. Sin embargo, la implementación del desenrollado la realizamos en Java.

La decisión de utilizar Java en lugar de C++ se debió, principalmente, a las siguientes consideraciones:

- Java es un lenguaje de más alto nivel que C++, por lo que ciertos aspectos —sobre todo, la gestión de memoria— pasaban a realizarse automáticamente.

- Si bien C++ es un lenguaje compilado y Java interpretado —con lo que los programas desarrollados en el segundo pueden ser notablemente más lentos—, la velocidad de ejecución del programa Java no era un aspecto

en absoluto relevante, ya que se trataba de generar un texto que, en uno u otro lenguaje, no llevaría más de unos segundos.

- Para generar el código en VHDL necesitábamos un lenguaje en el que se pudiera trabajar con cadenas de texto de la forma más sencilla posible. Java permite concatenar las cadenas de texto con todo tipo de datos —números, caracteres, otras cadenas de texto— utilizando un simple "+".

- Al generar el código VHDL es necesario almacenar el nombre de las señales que se utilizan para su posterior declaración. Las estructuras de datos necesarias para representar listas, diccionarios o conjuntos ya están presentes en el API de la edición estándar de Java y resultan muy sencillas de utilizar.

Ejemplo de metaprogramación

Un ejemplo de generación de un diseño en VHDL utilizando un programa en Java u otro lenguaje de alto nivel es el siguiente:

- Supongamos que tenemos un bucle en un programa en software que realiza el siguiente cálculo:

```
int total_a = 0;
int total_b = 0
for (int x = 0; x < 4; x++) {
    for (int y = 0; y < 4; y++) {
        total_a = total_a + valor[x+1][y];
        total_b = total_b + valor[y][x+1];
    }
}
```

- Podemos crear un hardware en VHDL que calcule los mismos valores que se estaban calculando anteriormente, sin necesidad de realizar varias iteraciones. Para ello, deberíamos obtener un hardware como el siguiente:

```
signal total_a, total_b : std_logic_vector(n - 1 downto 0);
begin
total_a <= valor10 + valor11 + ... + valor43;
total_b <= valor01 + valor11 + ... + valor34;
```

- Podría modificarse entonces el primer bucle para que, en lugar de realizar los cálculos, generase el código VHDL:

```
String total_a = "total_a <= ";
String total_b = "total_b <= ";
for (int x = 0; x < 4; x++) {
    for (int y = 0; y < 4; y++) {
        total_a += "+valor" + String(x+1) + String(y);
        total_b += "+valor" + String(y) + String(x+1);
    }
}
print(total_a);
print(total_b);
```

Utilizando esta técnica, implementamos de tres formas diferentes el hardware, cada una más optimizada que la anterior, aunque finalmente sólo utilizamos una de ellas. No obstante, ya que cada una se basa en la anterior, explicaremos todas.

6.2.1. Diseño directo

Inicialmente diseñamos de la forma más sencilla posible la red de cálculo. Para ello, implementamos en Java los bucles que en C++ utilizábamos para realizar el cálculo de los nodos de las submallas. Pero, en este caso, los bucles no realizan dicho cálculo sino que en cada vuelta se van escribiendo los valores y las señales necesarias para realizarlos. El resultado de las trazas inicialmente lo imprimíamos por consola, luego vimos útil crear un archivo **.vhd* para poder realizar cambios fácilmente en el código en caso de necesidad.

Para conocer los valores de Bernstein —las constantes que utilizamos en los productos— implementamos una función específica. Y no sólo eso, además, redujimos las multiplicaciones del código VHDL de la siguiente manera: cada sumando de los bucles anteriores es resultado de la multiplicación de un nodo y dos valores de Bernstein y, puesto que éstos últimos son fijos, incluimos el resultado de su multiplicación en el código en lugar de añadir otra operación más.

Además, se necesitan una serie de funciones auxiliares:

- Una función para convertir una de las constantes de Bernstein —dada en punto flotante— a una cadena de caracteres. Dicha cadena contiene la representación en texto del número binario codificado en punto fijo.

- Una función para implementar la multiplicación. Como ya vimos, calculamos una multiplicación como suma de desplazamientos. De esta manera, la función calcula el número de unos presentes en la constante por la que se multiplica al nodo y opera en consecuencia:
  - Si el número por el que se multiplica sólo tiene un bit a 1, se crea el código necesario para el desplazamiento —que lo asigna a una señal intermedia— y almacena el nombre de dicha señal.
  - Si el número por el que se multiplica tiene dos bits a 1, se crea el código para la suma de ambos desplazamientos y se almacena el nombre de la señal utilizada.
- Funciones para generar los nombres de las señales y el código necesario para realizar la declaración de señales en VHDL.

Utilizando la traza para desenrollar los bucles, obtuvimos expresiones en VHDL de la forma "`mallanNij <= mnijs1 + mnijs2 + ...`" donde:

- `mallanNij` es el nodo b_{ij} de la submalla n, donde $n \in \{0, \ldots, 3\}$.
- `mnijsk` es el sumando k-ésimo de la suma que produce el nodo `mallanNij`.

El problema de utilizar esta aproximación es que hay gran cantidad de desplazamientos que se realizan en múltiples ocasiones, además de haber también sumas enteras y algunas sumas parciales repetidas. La causa es que existen muchos nodos compartidos por las submallas. Por ello, el hardware resultante, aunque es totalmente funcional, es muy extenso debido a la gran cantidad de cálculos y señales "repetidos".

6.2.2. Diseño reducido

Con el objetivo de reducir el tamaño necesario para implementar el hardware, modificamos el código Java para que no generase VHDL donde hubiera distintas señales que calculasen los mismos valores. Para ello, basándonos en la aproximación anterior, realizamos las siguientes modificaciones:

- Creamos una clase `Desplazamiento` que representaba una señal que calculase el desplazamiento de un operando.
- Creamos también una clase `Suma`, que se componía de un conjunto de desplazamientos —los sumandos—.

Ahora, cuando el programa genera el código, opera de la siguiente forma:

- Para cada resultado, calcula el conjunto de sumandos que debe sumar.
- Si ya está almacenada una suma con ese mismo conjunto de sumandos[1], obtenemos el nombre de dicha señal y generamos el código `salidaX <= sumaY`.
- Si no hay una suma ya calculada que contenga todos los sumandos, se genera el código de la nueva suma.
    - Comprobamos, uno por uno, que existan todos los sumandos (desplazamientos). Para todos los que no existan, se crea una nueva señal y se genera el código que calcule dicho desplazamiento.

  Finalmente, se crea una señal a la que se le asigna la suma de todos los desplazamientos.

Utilizando esta técnica, obtenemos las siguientes ventajas:

- Cada desplazamiento se realiza una única vez.
- En todos los nodos que sean iguales, las salidas de los sumadores se reutilizan, en lugar de calcular varias veces lo mismo. El número de sumas que se realizaban más de una vez asciende a 13 (cada una de ellas de bastantes sumandos), como se puede ver en la sección 2.3.2 en todos los nodos que están remarcados.

Aunque de este modo el hardware se reduce bastante, aún quedaban dos posibles optimizaciones o mejoras que tratamos de realizar en una tercera aproximación.

6.2.3. Diseño reducido con sumadores en árbol

Si bien ya no hay salidas que tengan el mismo valor pero que se calcule varias veces, sí que sigue habiendo un cierto número de sumandos que pueden estar repetidos. Además, decidimos no dejar en manos del compilador de VHDL un aspecto importante: balancear los árboles de sumadores.

Para conseguir balancear los sumadores y reutilizar no sólo sumas totales sino también parciales, modificamos el algoritmo del siguiente modo:

[1]Cuando decimos "conjunto" nos referimos a un `HashSet`, un conjunto no ordenado de objetos. De este modo, como los sumandos son permutables, podemos ver que dos sumas sean iguales aunque sus sumandos estén en distinto orden.

- Cuando se desea generar el código necesario para una suma de varios sumandos, generamos recursivamente de forma explícita un árbol de sumadores:

  - Si el conjunto de sumandos tiene un sólo sumando, se devuelve directamente el nombre de la señal que contiene dicho desplazamiento. Al igual que con la solución anterior, si no existe una señal que contenga dicho desplazamiento, se crea tanto la señal como el código necesario para ello.
  - Si el conjunto de sumandos tiene más de un sumando, se divide en dos conjuntos de igual tamaño[2]. Después, recursivamente, se obtiene el nombre de la señal que contiene la suma de cada conjunto. Finalmente, se crea una señal a la que se asigna la suma de las dos señales anterior.

    Adicionalmente, si alguno de los dos subárboles de sumas ya existiera[3], no se crea el código, sino que se utiliza una señal existente. Como dicha señal estará balanceada, el árbol resultado sigue estando balanceado.

Aunque con esta modificación no se eliminaron apenas señales intermedias, nos aseguramos de que las sumas quedaran completamente balanceadas, independientemente de las optimizaciones que realizara el compilador de VHDL.

El listado de código correspondiente a estas 3 implementaciones puede verse en el apéndice C.

[2]Si el número de sumandos fuera impar, uno de los conjuntos tendría un sumando más. No obstante, el árbol resultante estará igualmente balanceado.

[3]Al comprobar si una suma existe, una vez más se realiza con un `HashSet`, de modo que se comparan los árboles de suma sin importar el orden de los sumandos.

Parte IV

Ensamblaje hardware-software

Capítulo 7

Desarrollo con FPGA

7.1. Introducción

Tras escribir el código VHDL correspondiente al hardware, procedimos a realizar una implementación sobre una FPGA *Xilinx Virtex 2 Pro*. Para ello, utilizamos el entorno de desarrollo *Xilinx EDK*[1] (*Embedded Development Kit*), un conjunto de programas que permiten realizar desarrollo hardware y software e integrarlo fácilmente. Concretamente, utilizamos la utilidad *XPS*, un entorno para la configuración de una FPGA que permite seleccionar distintos componentes y conectarlos de forma sencilla.

Dicho entorno nos permitió, además, compilar e integrar nuestro programa en el propio sistema, realizando mínimas modificaciones. Las herramientas software que proporciona XPS están basadas en el sistema operativo GNU[2], un sistema operativo tipo UNIX. Ya que nuestro software compilaba y ejecutaba sobre Linux, no fue difícil adaptarlo para que se pudiera ejecutar en la placa.

En cuanto al hardware, tuvimos que configurar un sistema a nuestro gusto, añadir un periférico que nos proporcionase algunas señales de GPIO y, finalmente, conectar dichas señales con el hardware que habíamos diseñado nosotros.

7.2. Modificaciones software

Nuestro primer objetivo fue hacer que el software que habíamos creado se ejecutase íntegramente en el procesador *PowerPC* de la placa, de modo

[1] `http://www.xilinx.com/ise/embedded_design_prod/platform_studio.htm`
[2] `http://www.gnu.org/`

que funcionase como si estuviese siendo ejecutado en un PC. Para ello, tuvimos que eliminar algunas funcionalidades que utilizamos para depuración —las trazas, conversiones de distintos elementos en cadenas de texto, etc— y modificar la función que utilizábamos para mostrar imágenes en la pantalla.

7.2.1. VGA

La representación de imágenes en pantalla nos hizo recurrir a la documentación de *Xilinx*. Primeramente, implementamos un ejemplo que nos permitiese mostrar unas bandas verticales de colores. Dicho código se puede ver en la sección F.1.

El acceso a VGA se realiza mediante escrituras en un *framebuffer* situado en una dirección de memoria conocida. Por defecto, la pantalla se encuentra configurada para mostrar una imagen de 640×480 pixels, a una frecuencia de refresco de 60Hz. Las direcciones de memoria del *framebuffer* crecen hacia arriba y la derecha de la imagen.

La rutina que creamos para poder dibujar puntos en pantalla puede verse en la sección B.7.2.

7.2.2. Acceso a hardware

Tras conseguir mostrar imágenes en un monitor, nuestro siguiente objetivo fue comprender cómo se realizaba el acceso a los pines de GPIO. Para ello, creamos un sistema que tenía 4 *switches* y 4 *leds* conectados a distintos GPIOs, e implementamos un programa que leyese el valor de los *switches* y lo mostrara en los *leds*. De este modo, aprendimos tanto a escribir como a leer valores, y además comprendimos qué rutinas eran necesarias para ajustar los buffers tri-estado.

El código de dicho ejemplo puede verse en la sección F.2.

7.2.3. Protocolo software

Una vez fuimos capaces de escribir y leer en los GPIOs, implementamos la parte software del protocolo. Vimos en la sección 5.4 la parte hardware, donde se describen las señales que intercambian el procesador y el controlador hardware para conseguir computar la subdivisión de mallas. Así, creamos un programa que ejecutaba literalmente el protocolo descrito, y cuya implementación se puede ver al final de la sección B.9.2.

Este programa se ejecutaba en sustitución de la función que calculaba los nodos de las submallas, de modo que el flujo del programa no cambia, salvo

porque los nodos de las submallas no se calculan llamando a una función auxiliar sino utilizando el hardware.

Finalmente, tuvimos que realizar algunas modificaciones al *script* de enlazado que permitía escoger en qué direcciones de memoria se almacenaban las distintas partes del programa: datos, instrucciones, pila de ejecución, *heap*. Almacenamos la totalidad del programa en la memoria dinámica, de forma que todo salvo el *heap* se almacenase en el primer banco de la memoria DDR, mientras que el *heap* se guardara en el segundo. De este modo, conseguimos que se pudieran utilizar hasta 256MBytes para el *heap* sin problemas.

7.3. Modificaciones hardware

Además de realizar modificaciones al programa principal para que pudiera realizar los cálculos en la unidad de hardware específico, también tuvimos que modificar nuestro propio diseño y el diseño del sistema para que pudieran conectarse.

7.3.1. GPIO

Modificamos el sistema que habíamos creado con XPS para que la placa incluyese una serie de pines GPIO. Para ello, hicimos que se integrase en la FPGA un *soft–core* que implementaba dos canales bidireccionales de 32 bits, que se conectaban con el procesador mediante el bus local del procesador (PLB). Aunque en un principio tratamos de usar el bus OPB, diseñado para periféricos con bajas necesidades de ancho de banda, finalmente hicimos pruebas con el bus PLB y observamos que nuestro hardware funcionaba más rápido.

Configuramos el periférico GPIO para que tuviese 64 pines agrupados en 2 canales bidireccionales. Uno de los canales sirvió como bus de datos, mientras que el otro se utilizó para las señales de control.

Aunque los pines son bidireccionales, nosotros establecimos las conexiones directamente con los registros de entrada y salida de los GPIO, de modo que no utilizamos los buffers tri-estado. Sin embargo, en la parte software, antes de realizar lecturas o escrituras, tuvimos que utilizar las primitivas de ajuste de dirección, para que el hardware funcionase de la forma adecuada.

Ajustamos la dirección base de los GPIO y el espacio de direcciones de memoria dedicado a ellos de modo que no interfiriese con otros periféricos. Después, añadimos nuestro propio dispositivo y conectamos sus señales de entrada y salida de control y datos con los pines de entrada y salida de ambos canales del GPIO.

7.3.2. Controlador

El controlador tuvo que ser modificado de modo que las señales de control que recibía y enviara estuvieran contenidas en dos buses, uno de entrada y otro de salida, de 32 bits de ancho. Escogimos qué pines del bus utilizaríamos como señales de control, y los conectamos a las entradas de control "auténticas" de nuestro hardware. Finalmente, el código VHDL utilizado es el que aparece en la sección D.1.

7.3.3. Integración

Finalmente, creamos la *netlist* y el *bitstream*. La *netlist* es una descripción de los elementos que componen el sistema y el interconexionado. El *bitstream* es una cadena de bits que contiene la información necesaria para que la FPGA implemente la *netlist*. Este proceso, con el primer diseño que realizamos, ocupó a un ordenador *AMD Athlon 64 3200+*, con 2Gbytes de memoria RAM, más de 12 horas de trabajo. Podemos observar el *floorplan*[3] en la figura 7.1.

7.4. Optimizaciones

Tras realizar la primera implementación, que gracias a la técnica utilizada para la creación del código VHDL funcionó desde el primer momento, decidimos tratar de reducir el tamaño del hardware. Ya que esta implementación utilizaba más del 75 % de los recursos de la placa, sería bastante difícil implementar aún más operaciones por hardware. Por ello, tratamos de eliminar todas las señales que realizaban cálculos "repetidos", como se puede ver en las secciones 6.2.2 y 6.2.3.

Podemos ver el desarrollo funcionando en la figura 7.2.

7.4.1. Problemas encontrados

Problemas de diseño

El problema de utilizar las implementaciones optimizadas es que, al disminuir la cantidad de señales que contienen cálculos repetidos, aumenta el *fanout* —la cantidad de lógica a la que se conecta una misma señal—. Además, a pesar de que la latencia de la red de cálculo disminuye, *Xilinx Platform Studio* informa de que no puede cumplir las restricciones de distribución de

[3]El *floorplan* es una imagen de la utilización e interconexionado de las distintas celdas de las que se compone una FPGA.

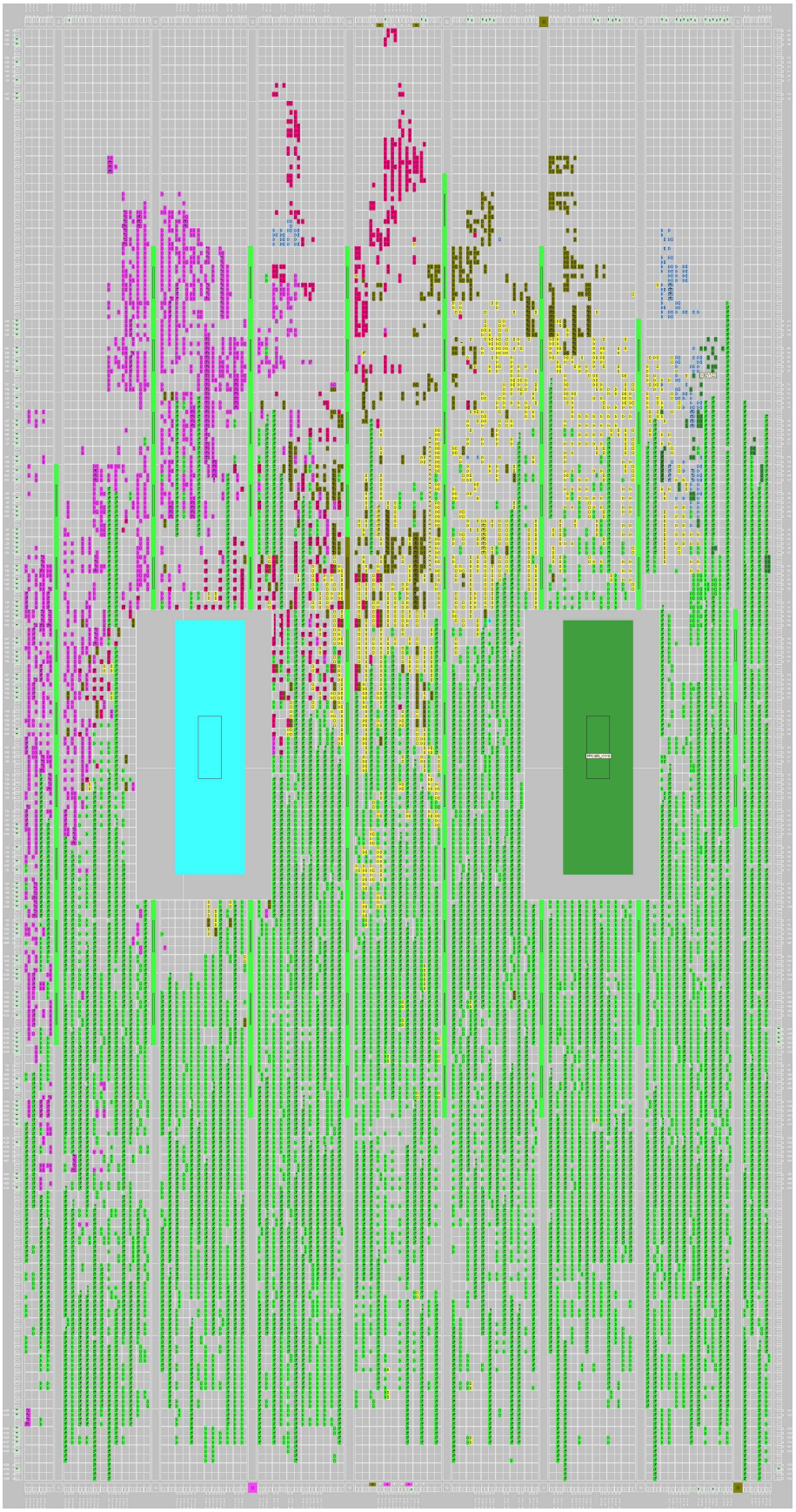


Figura 7.1: Floorplan.

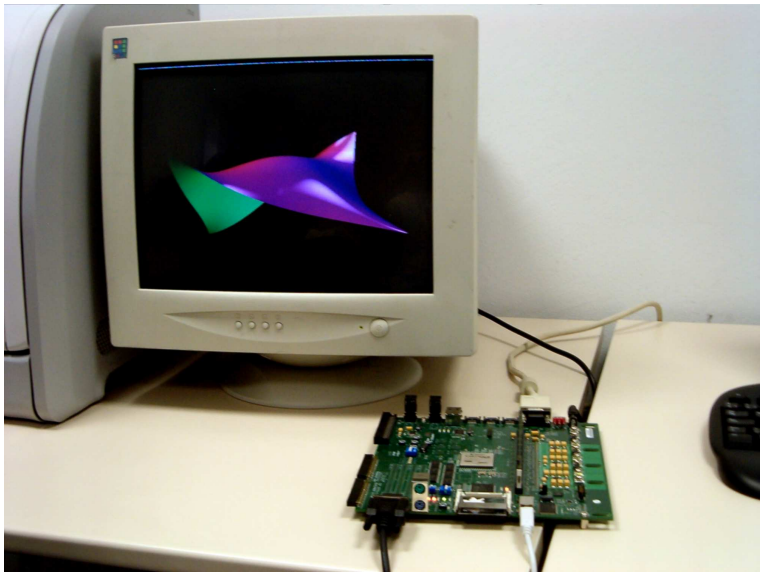


Figura 7.2: Placa Virtex II Pro ejecutando el programa sobre el hardware diseñado.

la señal de reloj, y no permite implementar el diseño. Conseguimos resolver el problema de la temporización introduciendo un *latch* a la salida del multiplexor final de nuestro hardware, tanto en la primera optimización como en la optimización con el diseño de sumadores en árbol.

En ambos casos, aplicando esta técnica conseguimos que la herramienta sintetizase el circuito. Además, redujimos enormemente la cantidad de hardware necesario, disminuyendo casi a la mitad los recursos utilizados. Sin embargo, en ambos casos apareció un problema que no fuimos capaces de resolver: al escribir en algunas ocasiones un valor en los registros de control de la lógica tri-estado, el procesador se bloquea y no responde al depurador, de modo que hemos sino incapaces de saber a qué estado llega.

Creemos, sin embargo, que el problema puede estar, además de en la distribución de reloj, en que algunas señales tienen un *fanout* elevadísimo, llegando a conectarse una misma salida a 48 entradas, por lo que ciertas operaciones sobre el bus PLB hacen que se bloquee el procesador.

Otros problemas

También nos encontramos con otros problemas, sobre todo por desconocimiento de la herramienta de desarrollo. En concreto, aunque creamos un periférico y lo añadimos en el sistema que habíamos creado con el *EDK*, no fuimos capaces de que éste utilizase otras entidades VHDL definidas en va-

rios archivos. Tampoco nos aceptó todas las entidades en un mismo archivo VHDL, sino que tuvimos que mezclar todo el código para que formase parte de una única entidad, y utilizar ese archivo como parte del sistema final. Sabemos que es por desconocimiento de la herramienta y no del lenguaje VHDL porque con otras herramientas de *Xilinx* —en concreto con *Xilinx ISE*— sí pudimos llegar a sintetizar, e incluso simular utilizando *ModelSim*.

También nos costó algunas semanas conseguir ejecutar código en la placa, ya que la utilización del depurador y las herramientas de programación de la placa no siempre producen códigos de error cuyo significado sea fácilmente interpretable.

7.5. Comparativa

Realizamos una comparación entre el cálculo por hardware y el cálculo software. Para ello, realizamos la división de una malla una serie de veces utilizando los dos métodos, y comparamos el número de ciclos de reloj que transcurrían hasta terminar el cálculo. Además, para ser más justos en la comparativa, realizamos dos ajustes:

- En el cálculo software, realizamos el cálculo en punto fijo en lugar de en punto flotante. Realizamos esta modificación porque el *PowerPC 405* no tiene unidad de punto flotante, y realiza tales cálculos en modo software.

- En el cálculo hardware, para tratar de paliar el cuello de botella que supone el uso de los GPIOs —que retrasa varios ciclos cada escritura o lectura—, implementamos nuestras propias funciones de acceso a hardware. Para ello, en lugar de utilizar las primitivas de lectura, escritura y ajuste de los buffers tri-estado que proporciona *Xilinx*, realizamos directamente las escrituras necesarias en las direcciones de memoria adecuadas. De este modo, nos ahorramos todas las comprobaciones que hacen las rutinas de *Xilinx* y además ahorramos algunas instrucciones más al no realizar llamadas a funciones (paso de parámetros, cambio de contexto, saltos, etc). Sin embargo, lo que no somos capaces de reducir es la cantidad de ciclos que se necesitan para realizar una operación en el bus PLB.

Con todo, el número de ciclos necesarios es el siguiente:

Iteraciones	Ciclos para SW	Ciclos para HW
10	213301	51727
100	2128211	508011
1000	21282011	5092904
10000	212820011	50902710
100000	2128200011	509173260
1000000	21282000011	5091716582

La media de ciclos que toma cada iteración es de aproximadamente 21.000 en el caso software, y aproximadamente 5000 en el modo hardware. Sin embargo, cada ciclo de escritura/lectura en el caso hardware se podría realizar con sólo 3 ciclos, con lo que el mínimo número teórico de ciclos es de

$$3 \cdot (16 + 64) = 240$$

Esta cifra es mucho más baja que los 5000 ciclos de media que está llevando, y se debe a la lenta comunicación entre el procesador y el bus al que se conecta nuestro hardware.

Como vemos, actualmente el cálculo por hardware es unas 4 veces más rápido que el cálculo software, aunque si pudiésemos llevar el hardware al límite, sería de unas $21000/240 = 87.5$, que como vemos es una mejora de casi 2 órdenes de magnitud respecto al cálculo por software.

El listado de código del programa utilizado para comparar los tiempos está en la sección F.3.

Capítulo 8

Conclusiones

8.1. Objetivo inicial

Cuando comenzamos este proyecto, nuestro propósito era desarrollar un sistema de representación de superficies basado en splines y no en triángulos, y en el que el límite a la calidad de las imágenes no fuese la definición de los objetos a mostrar, sino la pantalla sobre la que se mostrase.

Además, nuestra intención en cuanto a la implementación hardware fue la de encontrar algún tipo de diseño que nos permitiese mejorar los tiempos de ejecución del algoritmo, a la par que demostrar que podía crearse una arquitectura especializada.

8.2. Resultados obtenidos

Si comparamos nuestros objetivos iniciales con los resultados obtenidos, se puede decir que los hemos cumplido. Hemos conseguido un sistema funcional, capaz de crear imágenes sintéticas partiendo de una cantidad de información inicial mucho menor que en los sistemas actuales —esto es, que la cantidad de información necesaria para codificar la forma de los objetos que representamos es mucho menor que en un sistema basado en mallas de triángulos—. Además, también hemos desarrollado hardware para acelerar la operación más lenta del algoritmo, y hemos obtenido los conocimientos suficientes para implementar el necesario para acelerar la ejecución del algoritmo.

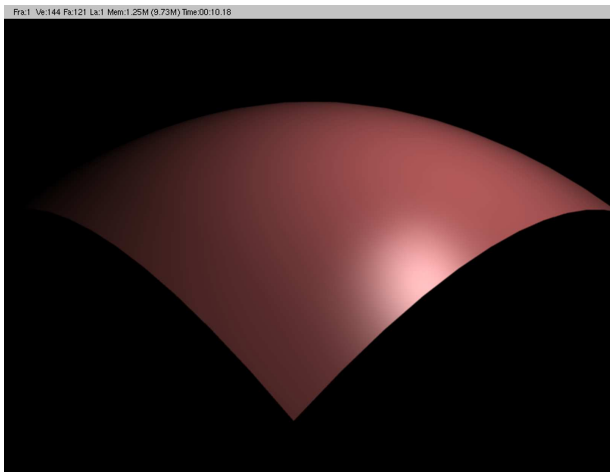


Figura 8.1: Representación de una superficie NURBS en *Blender*.

8.3. Comparativa con los sistemas actuales

Lo primero que hay que tener en cuenta es que nuestro sistema difiere totalmente de los sistemas de representación tradicionales basados en triángulos, por lo que no es sencillo realizar una comparación entre ambos sistemas. Además, las matemáticas y los algoritmos utilizados en la representación basada en triángulos llevan poniéndose en práctica y mejorando desde principios de la década de 1970, mientras que nuestro sistema es el producto de menos de un año de desarrollo. Por lo tanto, no podemos realizar más que una comparación teórica.

8.3.1. Tiempo de computación

Para realizar una comparativa del tiempo de ejecución con los sistemas basados en triángulos hemos utilizado *Blender*[1]. Se trata de un programa de diseño gráfico que permite dibujar NURBS[2]. Hemos elegido las NURBS porque el programa no permite representar splines, así que utilizamos este tipo de superficies que son las más similares a las que utilizamos nosotros. La imagen generada se puede ver en la figura 8.1.

Creamos una superficie NURBS sencilla para hacer la prueba. Este programa tarda en crear la imagen un tiempo aproximado de 11 segundos. Midiendo después el tiempo de ejecución de la implementación software de nues-

[1] `http://www.blender.org`

[2] Superficies matemáticamente similares a las splines, pero que permite un control de la forma más preciso y, por lo tanto, más expresivo.

tro programa, éste tarda un tiempo aproximado de 10 segundos.

Aunque el tiempo de ejecución de nuestro programa es menor, no es una gran diferencia. Sin embargo, hay que tener en cuenta que *Blender* es un programa que lleva muchos años de desarrollo y que, actualmente,se usa de forma profesional en el mundo de la animación. Se ha utilizado, incluso, para realizar la previsualización de escenas en Spiderman 2 y ha colaborado en la realización algunos cortometrajes. Por lo tanto, esto le otorga cierta ventaja a nuestro trabajo, ya que en unos meses hemos conseguido un tiempo de ejecución similar a un programa basado en el método tradicional de mallas de triángulos.

8.3.2. Uso de memoria

Una clara ventaja de las imágenes definidas mediante curvas polinómicas, es que se necesita mucha menos memoria para almacenarlas que si están definidas mediante mallas de triángulos. Esto es debido a que, según explicamos en **??**, para representar cualquier imagen con superficies curvas mediante mallas de triángulos, se necesita una gran cantidad de vértices, mientras que usando curvas polinómicas solo usamos 16 vértices por cada superficie spline en que se divide la figura.

Sin embargo, nuestra implementación tiene una desventaja con respecto al sistema tradicional. Nosotros necesitamos usar mucha más memoria intermedia debido a la técnica de programación dinámica —guardamos las submallas de control para reducir el tiempo de ejecución—, mientras que en el sistema de división en mallas de triángulos el uso de memoria intermedia es despreciable.

8.3.3. Precisión

Como ya explicábamos en la introducción, la representación basada en splines produce imágenes de superficies curvas cuya precisión sólo se ve limitada por la resolución de la pantalla donde ésta se muestre. Sin embargo, en programas como *Blender*, para representar NURBS primero se realizar una conversión a una malla de triángulos, por lo que, aunque existen técnicas para representar superficies polinómicas con el sistema de representación actual, siguen estando limitadas por el hecho de utilizar triángulos en última instancia.

8.4. Ampliaciones del proyecto

Como posibles ampliaciones del proyecto, hemos identificado una serie de aspectos en los que se podría mejorar el trabajo que hemos realizado. Se pueden separar en 2 grupos: mejoras conceptuales —funcionalidades que se podrían añadir al algoritmo— y mejoras de implementación —técnicas de programación y técnicas hardware que acelerarían el proceso—.

8.4.1. Mejoras del algoritmo

Texturas

Una de las capacidades de las que carece nuestro algoritmo es la representación de objetos con texturas. Sin embargo, por la forma en que trabaja el algoritmo, resulta muy sencillo saber qué punto de una textura se corresponde con un punto de la imagen en que se ve un objeto.

Recordemos que una spline es una superficie que se interpola en 2 direcciones (u y v) a partir de una serie de puntos. Dicha superficie se define como una función

$$\varphi : [0,1] \times [0,1] \rightarrow \mathbb{R}^3$$

Es decir, se trata de una función que, a partir de dos valores $u, v \in [0,1]$, genera un conjunto de puntos en el espacio que definen una superficie.

Ahora bien, si hacemos corresponder las "esquinas" de la spline[3] con las esquinas de la imagen que se utiliza como textura, para saber qué punto de la textura se corresponde con un punto de la spline sólo es necesario saber los valores de u y de v para la superficie en dicho punto.

Para calcularlos, bastaría con realizar una ampliación mínima del algoritmo. Sea la superficie inicial φ definida anteriormente, cuando se aplica la operación de división se obtienen 4 superficies que se describen matemáticamente con el mismo polinomio que la spline original, salvo un cambio de variable. Por lo tanto, si dividimos la spline original utilizando el valor $t = 0{,}5$ al evaluar los polinomios de Bernstein, las 4 superficies que se obtienen son:

$$\varphi : [0,1] \times [0,1] \rightarrow \mathbb{R}^3 \Rightarrow \begin{cases} \varphi_1 : [0,0.5] \times [0,0.5] \rightarrow \mathbb{R}^3 \\ \varphi_2 : [0,0.5] \times [0.5,1] \rightarrow \mathbb{R}^3 \\ \varphi_3 : [0.5,1] \times [0,0.5] \rightarrow \mathbb{R}^3 \\ \varphi_4 : [0.5,1] \times [0.5,1] \rightarrow \mathbb{R}^3 \end{cases}$$

Dicho de forma más general, sea una superficie cualquiera interpolada en el intervalo $[a,b] \times [a,b]$, donde, en el caso de ser una superficie dividida 0

[3]Las esquinas de la malla son aquellos puntos del espacio en los que u y v valen 1 ó 0, es decir, $(u,v) = (0,0)$, $(u,v) = (0,1)$, $(u,v) = (1,0)$ y $(u,v) = (1,1)$.

o más veces, $a, b \in [0,1]$, entonces si dividimos la superficie, las superficies resultantes son:

$$\varphi : [a,b] \times [a,b] \to \mathbb{R}^3 \Rightarrow \begin{cases} \varphi_1 : \left[a, \frac{a+b}{2}\right] \times \left[a, \frac{a+b}{2}\right] \to \mathbb{R}^3 \\ \varphi_2 : \left[a, \frac{a+b}{2}\right] \times \left[\frac{a+b}{2}, b\right] \to \mathbb{R}^3 \\ \varphi_3 : \left[\frac{a+b}{2}, b\right] \times \left[a, \frac{a+b}{2}\right] \to \mathbb{R}^3 \\ \varphi_4 : \left[\frac{a+b}{2}, b\right] \times \left[\frac{a+b}{2}, b\right] \to \mathbb{R}^3 \end{cases}$$

Si suponemos que, cuando una superficie cabe en una caja del tamaño de un pixel, podemos aproximar el centro de la spline por el centro de la caja, podríamos aproximar el valor de los parámetros en el punto central de la spline por $(a/2, b/2)$. Con otras palabras, bastaría con que la spline inicial conociese sus valores máximo y mínimo de u y de v, y que al dividir cada submalla calculase los suyos a partir de la malla original. Finalmente, el punto de la textura que se corresponde con el punto de la spline es

$$\left(\frac{u_{max} + u_{min}}{2}, \frac{v_{max} + v_{min}}{2}\right)$$

teniendo en cuenta que la textura se mueve en el intervalo $[0,1] \times [0,1]$.

Materiales

Al igual que en el caso anterior, el algoritmo no es capaz de simular distintos materiales, pero las modificaciones necesarias para hacerlo son también mínimas. Bastaría con parametrizar las cantidades de luz difusa y especular que refleja un objeto para simular distintos materiales, y podría realizarse un *bump-mapping*[4] para simular rugosidades.

[4]La técnica del *bump-mapping* consiste en utilizar una imagen para dar rugosidad a una superficie. Para ello, se actúa como si se fuera a representar una textura, salvo que, a la hora de mostrar la imagen, no se pondera la cantidad de luz reflejada con el color de la textura en ese punto, sino que lo que se hace es modificar la normal de la superficie ligeramente —en función del color de la textura en ese punto y en los puntos vecinos, es decir, del gradiente de la imagen que se use como textura en dicho punto— para realizar los cálculos de luz especular y difusa. De este modo, se consigue un efecto computacionalmente "barato" y bastante realista para objetos que no tengan una rugosidad muy acusada, tal como paredes, gravilla, telas...

Iluminación

La implementación que hemos realizado del algoritmo permite trabajar con un número ilimitado de fuentes de luz, pero todas ellas son focos omnidireccionales. Sin embargo, las técnicas de representación basadas en mallas de triángulos utilizan otros tipos de iluminación que también serían implementables en nuestro algoritmo. Tales tipos de luz serían:

- Luz "de ambiente". Se trata de un tipo de luz que proviene de todas direcciones. Se podría implementar sumando un determinado valor a todos los pixels cuya linea de visión corta con una spline.

- Luces direccionales. Este tipo de iluminación se diferencia de la nuestra en que la luz, en lugar de provenir de un foco, proviene "del infinito". Esto hace que los rayos de luz de esta fuente sean todos paralelos. Se podría implementar haciendo que el vector que une el punto con la luz no se calcule cada vez, sino que fuese constante.

- Luces "de foco", o luces que sólo iluminan los objetos que están dentro de su cono de luz. Su implementación se podría realizar en función de dos vectores: el vector que une el punto con el foco y el vector dirección de la luz. De este modo, si ambos vectores forman un ángulo mayor que un α dado, la luz no ilumina.

Sombras y *raytracing*

La representación de sombras se podría realizar mediante dos técnicas: mapas de sombras o *raytracing*.

Los mapas de sombras se basan en una técnica en dos pasos. Primero, se calcula qué partes de la imagen son visibles desde el punto de vista de la luz, y después se dibuja la imagen normalmente, pero teniendo en cuenta qué puntos deberán estar iluminados y cuáles no. Aunque es una técnica muy popular hoy en día porque permite utilizar hardware existente en las tarjetas gráficas, en nuestro algoritmo podría no ser tan sencilla de implementar.

La otra técnica, el *raytracing*, consiste en unir el punto de corte que estamos dibujando con la fuente de luz. Después, habría que calcular si dicha recta corta con alguna spline. Si efectivamente corta, entonces esa luz no ilumina el punto y se vería una sombra, y en caso contrario habría que calcular la cantidad de iluminación —difusa, especular, etc— que el objeto hace llegar al observador.

La ventaja de utilizar el raytracing es que se trata de una técnica mucho más avanzada que los mapas de sombra, en el sentido de la calidad de las

sombras que produce. Además, con este algoritmo y las sombras por *raytracing* ocurre como con los mapas de sombra y la técnica de representación basada en mallas: los cálculos necesarios para las sombras se pueden ejecutar sobre el propio hardware existente.

Por otro lado, la técnica del *raytracing* nos podría permitir dibujar cierto tipo de materiales que, de otra manera, resultaría mucho más complicado. Por ejemplo, podríamos dibujar materiales reflectantes como el metal, o materiales refractantes como el cristal. Tan sólo hay que ejecutar más veces el algoritmo de corte entre una recta y un spline para conseguir el resultado adecuado.

NURBS

Una de las mayores carencias del algoritmo es la imposibilidad de trabajar con *NURBS*. Estos objetos son un tipo especial de splines que se diferencian de las que hemos utilizado en dos características esenciales: son invariantes por proyección —puede aplicarse una función de proyección a una NURBS y el resultado sigue siendo una NURBS— y además sus vértices tienen "peso". Este peso nos permite modificar la influencia que tiene la posición de cada vértice de la malla de control en la superficie generada.

Estas dos características son muy deseables en diseño gráfico, pues dotan a los artistas de una herramienta mucho más expresiva que las splines convencionales. No obstante, nuestro algoritmo apenas necesita ser modificado para representar NURBS: el algoritmo de división no cambia en líneas generales, salvo que los nodos, en lugar de tener 3 componentes (x, y, z), tendrían cuatro componentes (x, y, z, w), donde w es el peso de cada vértice. Ya que la división se realiza componente a componente[5], sólo habría que realizar una iteración más en la división para poder representar NURBS.

Antialiasing

Otra posible ampliación del algoritmo sería aplicar la técnica del antialiasing. El *aliasing* son los pequeños "escalones" que aparecen en pantalla cuando se dibujan curvas o líneas oblicuas. La mayor parte de las tarjetas gráficas modernas implementan algunas soluciones para tratar de eliminar estos efectos. Una de las más comunes es el FSAA o *Full Scene AntiAliasing*,

[5]El algoritmo de división de splines divide puntos, no componentes. Esto es, que aplica las operaciones matemáticas a cada componente de los vértices de forma independiente. En nuestro hardware actual, primero se procesan las x, después todas las componentes y y, finalmente, las z. Para el caso de las NURBS, sólo habría que hacerlo una cuarta vez para procesar los pesos.

una técnica que se llama así porque consiste en eliminar el *aliasing* tratando toda la escena en lugar de sólo las zonas afectadas. Para ello, lo que se hace es representar la imagen 2, 4, 8, y hasta 32 veces más grande del tamaño original, para luego aplicar un algoritmo que la "encoge", quedando difuminados los "escalones".

Para evitar tener que calcular el dibujo de una imagen 2 ó 4 veces y después "encogerla", lo cual resulta en una cantidad de trabajo bastante grande, una posible técnica podría ser:

- Computar la imagen normalmente, pero almacenando la "profundidad" a la que está cada pixel.

- En todos aquellos pixels que hagan frontera con otro cuya profundidad supere un cierto umbral, se realiza un dibujo ampliado local y, después, se "encoge" ese trozo de imagen.

Otra técnica similar ya se utiliza en algunas consolas, como la plataforma *XBOX*, que aplica el *antialiasing* sólo a los bordes de los objetos.

8.4.2. Mejoras de implementación

Además de las ampliaciones conceptuales, hemos identificado una serie de puntos en la implementación que podrían ser mejorados de cara a obtener un mayor rendimiento.

Cálculo en punto flotante

Uno de nuestros principales problemas ha sido el de no disponer de hardware de punto flotante ni de tiempo para diseñarlo. Para solventar este problema, decidimos limitar la malla de control de la spline a dibujar dentro de unas dimensiones conocidas, de modo que pudiésemos operar con aritmética de punto fijo.

Aunque trabajar con esta aritmética es mucho más rápido en principio —ya que podemos realizar operaciones aritméticas de forma combinacional—, tiene dos importantes desventajas:

- El rango de los resultados de las operaciones debe ser conocido de antemano.

- Ciertas operaciones en el algoritmo se sabe que no están dentro del rango. Por lo tanto, tales cálculos debemos efectuarlos en punto flotante.

Así pues, sería muy beneficioso para el rendimiento del algoritmo poder trabajar en punto flotante sin tener que realizar las conversiones —que por otro lado pueden llegar a ser muy costosas—, o bien disponer de un hardware capaz de transformar los datos de punto flotante a punto fijo y viceversa con un coste en tiempo menor que el actual.

Una posible solución para pasar de punto fijo a flotante podría ser combinacional. Resultaría tremendamente rápida, pero podría llegar a ocupar una gran área de integración. Para ello, habría que distinguir casos en función de qué bit a 1 del número en punto fijo es el más significativo. Si se tratase del primer bit, habría que copiar los primeros bits del número en punto fijo a la mantisa del número en punto flotante, y poner como exponente un valor precalculado. Si se tratase del segundo bit, se copiarían a partir del segundo a la mantisa y se pondría un exponente que sería el valor precalculado anterior menos 1, y así sucesivamente. Además, habría que distinguir los casos especiales del 1 y del 0 decimales, pero de podría conseguir implementar tal diseño utilizando sólo un codificador de prioridad y un multiplexor 32 a 1 de 32 bits de ancho, en el caso del punto fijo de 32 bits.

Obtención de la caja contenedora

Otra forma de reducir el tiempo de ejecución del algoritmo utilizando hardware específico, sería mediante la construcción de la caja contenedora por hardware. Actualmente, para construir la caja lo que hacemos es identificar dos esquinas opuestas. Para ello, comparamos todos los puntos entre sí para quedarnos con las coordenadas (x, y, z) numéricamente más bajas y más altas. Ya que la caja se construye en el momento de construir la spline, se podría modificar el hardware de división para que devolviese las coordenadas de la caja contenedora de cada submalla. Las comparaciones que sería necesario realizar para obtener el valor más alto y más bajo de los 16 valores de salida de cada malla resultado, se podrían realizar con un árbol de comparadores de 4 niveles, que podría calcular el resultado en aproximadamente un ciclo de reloj.

Corte de la recta y la caja

Aunque la operación de división de mallas es la que más operaciones y tiempo de cálculo requiere, la operación que más veces se ejecuta es la comprobación de corte entre una recta y una caja. Por ello, es una de las operaciones que más efectivo resultaría implementar por hardware, siempre que se dispusiera de un interfaz entre el procesador y dicho hardware con

muy baja latencia[6].

El problema del corte entre recta y caja es que dan lugar a divisiones que pueden producir resultados con magnitudes muy dispares, por lo que no se podría recurrir al punto fijo. Además, una de las ventajas de utilizar este sistema es que se podrían obtener los parámetros[7] λ en paralelo, y realizar las comparaciones utilizando un árbol, por lo que se podría reducir el tiempo necesario para el cálculo de 4 a 6 veces[8].

Paralelización

Finalmente, una de las mejoras en la implementación que podría reducir más el coste en tiempo sería calcular concurrentemente más de un pixel. Para ello, sería necesario disponer de tantos procesadores y tanto hardware específico como pixels se quisieran calcular a la vez. Además, se podría seguir aplicando la técnica de programación dinámica que estamos utilizando ahora mismo[9], de modo que los procesadores compartiesen un árbol de mallas en memoria. La lectura del árbol para llegar hasta la hoja que hubiera que dividir no sería un problema, pero habría sincronizar los procesadores para que en el momento de generar las submallas no realizasen la misma división.

[6]Como vimos en la sección 7.5, es muy importante que el interfaz que se utilice para la comunicación con el hardware sea muy rápido. En el caso del cálculo de las submallas, la cantidad de operaciones a realizar sobre los datos de entrada es tan grande que merece la pena trabajar en hardware a pesar de la lentitud del interfaz utilizado. Sin embargo, en una operación como la que nos ocupa, el reducido número de operaciones a realizar aconseja realizar una implementación hardware sólo si el interfaz que se utilice tiene una latencia mínima.

[7]Véase la sección 2.4.2.

[8]En la implementación que hemos realizado del corte entre recta y caja, realizamos un total de 4 divisiones, que son las operaciones más costosas. Por lo tanto, el tiempo de cálculo se reduciría hasta casi el tiempo necesario para realizar las divisiones, más las comparaciones. Sin embargo, nosotros sólo tenemos 4 divisiones porque sólo trabajamos con unas rectas muy concretas, que tienen una pendiente conocida en el eje z. Si nuestras rectas pudiesen estar en cualquier posición (como habría que realizar para poder aplicar las técnicas de sombreado y *raytracing*, por ejemplo) entonces tendríamos 6 divisiones, y el tiempo de cálculo en hardware sería casi 6 veces menor que en software.

[9]Véase la sección 4.4.1.

Parte V

Apéndices

Apéndice A

Implementación en Maple

El presente apéndice contiene el listado de código en Maple que contiene la primera implementación del algoritmo y las comprobaciones de su funcionamiento.

A.1. Inicializaciones

```
Digits := 40:
with(plots):
with(LinearAlgebra):
```

A.2. Bernstein

A.2.1. Cálculo del polinomio de Bernstein

```
Bernstein:= proc (n,i)
   option remember;
   if ((n=0)and(i=0)) then
      return 1;
   else
      if ((i>n)or(i<0)) then return 0;
      else
         return (1-'t') * Bernstein (n-1,i)+ 't' *
                          Bernstein (n-1,i-1);
      fi;
   fi;
end proc:
```

A.2.2. Polinomios de Bernstein con n:=3

```
DibujaBernstein := proc()
option remember;
local polinomios, i;

    polinomios:=[];

    for i from 0 to 3 do
        polinomios:=[op(polinomios),Bernstein(3,i)];
    od;
    return plot(polinomios, 't'=0..1);
end proc:
```

A.3. Spline

A.3.1. Malla de control

```
MallaDeControl := [
    [[0,1,1], [1,1,1], [2,1,1], [3,1,1]],
    [[0,2,1], [1,2,2], [2,2,2], [3,2,1]],
    [[0,3,1], [1,3,2], [2,3,3], [3,3,1]],
    [[0,4,1], [1,4,1], [2,4,1], [3,4,1]]]:
MallaDeControl2 := [
    [[3,1,1],[4,1,1],[5,1,1],[6,1,1]],
    [[3,2,1],[4,2,0],[5,2,-1],[6,2,1]],
    [[3,3,1],[4,3,0],[5,3,2],[6,3,1]],
    [[3,4,1],[4,4,1],[5,4,1],[6,4,1]]]:
MallaDeControl3 := [
    [[-1,-1,3.5],[0,-1,3.75],[1,-1,3.75],[2,-1,3.5]],
    [[-1,0,3.25],[0,0,2],[1,0,2],[2,0,3.25]],
    [[-1,1,2.75],[0,1,1.5],[1,1,1.5],[2,1,2.75]],
    [[-1,2,2],[0,2,2.25],[1,2,2.25],[2,2,2]]]:
```

A.3.2. Dibujo de malla de control

```
dibujaPoligonoControl := proc(mc, color)
option remember;
  local i,j,puntos;
  puntos:=[];
  for i from 1 to 3 do
```

```
    for j from 1 to 3 do
      puntos:=[op(puntos),[mc[i][j],mc[i][j+1],
                           mc[i+1][j+1],mc[i+1][j]]];
    od;
  od;
  return PLOT3D(POLYGONS(puntos[]), STYLE(WIREFRAME),
                color);
end proc:
```

Ejemplo de uso

```
dibujaPoligonoControl(MallaDeControl,
                      COLOUR(RGB,1,0,0));
dibujaPoligonoControl(MallaDeControl2,
                      COLOUR(RGB,0,1,0));
```

A.3.3. Cálculo de un punto de la Spline

```
calculaPuntoSpline:= proc(malla,u,v)
  local i, j, B1, B2, punto;
 option remember;
  punto:=[0,0,0];
  for i from 1 to 4 do
    for j from 1 to 4 do
      B1 := Bernstein(3,i-1);
      B1 := subs({'t'=u},B1);
      B2 := Bernstein(3,j-1);
      B2 := subs({'t'=v},B2);
      punto:= punto + malla[i][j] * B1 * B2;
    od;
  od;
return(punto);
end proc:
```

Ejemplo de uso

```
calculaPuntoSpline(MallaDeControl,0.5,0.5);
calculaPuntoSpline(MallaDeControl2, 0.5, 0.5);
```

A.3.4. Dibujo de la Spline

```
pintaSpline:= proc(malla, color)
   option remember;
```

```
    return plot3d([
        calculaPuntoSpline(malla,u,v)[1],
        calculaPuntoSpline(malla,u,v)[2],
        calculaPuntoSpline(malla,u,v)[3]],
        'u'=0..1, 'v'=0..1);
end proc:
```

Ejemplo de uso

```
pintaSpline(MallaDeControl, COLOUR(RGB,1,0,0));
pintaSpline(MallaDeControl2, COLOUR(RGB,0,0,1));
```

A.4. Nodos intermedios

A.4.1. Cálculo de los nodos intermedios de la Spline

```
calculaNodoIntermedio:= proc (malla,i,j,r,s)
local punto,k,l, B1, B2;
option remember;
punto:=[0,0,0];
for k from 1 to r+1 do
    for l from 1 to s+1 do
        B1 := subs({t=0.5},Bernstein(r,k-1));
        B2 := subs({t=0.5},Bernstein(s,l-1));
        punto:= punto + malla [i+k][j+l] * B1 * B2;
    od;
od;
return punto;
end proc:
```

A.4.2. Prueba: el punto intermedio es el mismo que $u = 0{,}5$ y $v = 0{,}5$

```
evalb(
    calculaNodoIntermedio(MallaDeControl,0,0,3,3) =
    calculaPuntoSpline(MallaDeControl, 0.5, 0.5)
);
```

A.5. Subdivisión de superficies

A.5.1. Cálculo de los sub-parches

```
subdivideMalla := proc(malla)
option remember;
local aux1, aux2, aux3, aux4, nPuntos1, nPuntos2,
      nPuntos3, nPuntos4, i, j;
#Abajo izquierda
nPuntos1 := []:
nPuntos2 := []:
nPuntos3 := []:
nPuntos4 := []:
for j from 0 to 3 do
   aux1 := [];
   aux2 := [];
   aux3 := [];
   aux4 := [];
   for i from 0 to 3 do

      #Abajo izquierda
      aux1 := [op(aux1),
               calculaNodoIntermedio(malla,0,0,j,i)];

      #Arriba izquierda
      aux2 := [op(aux2),
            calculaNodoIntermedio(malla,0, i, j, 3-i)];

      #Abajo derecha
      aux3 := [op(aux3),
             calculaNodoIntermedio(malla,j, 0, 3-j,i)];

      #Arriba derecha
      aux4 := [op(aux4),
           calculaNodoIntermedio(malla,j, i, 3-j,3-i)];
   od;
   nPuntos1 := [op(nPuntos1), aux1];
   nPuntos2 := [op(nPuntos2), aux2];
   nPuntos3 := [op(nPuntos3), aux3];
   nPuntos4 := [op(nPuntos4), aux4];
od:
return [nPuntos1, nPuntos2, nPuntos3, nPuntos4];
end proc:
```

A.5.2. Parche y subparches: dibujo de las mallas de control

```
dibujaMallas:= proc (Malla, subMallas)
option remember;

display(
    dibujaPoligonoControl(subMallas[1],
                          COLOUR(RGB,0,1,0)),
    dibujaPoligonoControl(subMallas[2],
                          COLOUR(RGB,0,0,1)),
    dibujaPoligonoControl(subMallas[3],
                          COLOUR(RGB,1,1,0)),
    dibujaPoligonoControl(subMallas[4],
                          COLOUR(RGB,0,0,0)),
    dibujaPoligonoControl(Malla,
                          COLOUR(RGB,1,0,0))
);
end proc:
dibujaMallas(MallaDeControl,
             subdivideMalla(MallaDeControl));
dibujaMallas(MallaDeControl2,
             subdivideMalla(MallaDeControl2));
```

A.5.3. Dibujo de una malla y sus submallas

```
pintaHijos:= proc(malla)
option remember;
    local subMallas;

    #Calculamos las submallas
    subMallas := subdivideMalla(malla);

    display3d(
        plot3d([calculaPuntoSpline(malla,u,v)[1],
            calculaPuntoSpline(malla,u,v)[2],
            calculaPuntoSpline(malla,u,v)[3]],
            'u'=0..1,'v'=0..1,color=yellow),
        plot3d([calculaPuntoSpline(subMallas[1],u,v)[1],
            calculaPuntoSpline(subMallas[1],u,v)[2],
            calculaPuntoSpline(subMallas[1],u,v)[3]],
```

```
        'u'=0..1,'v'=0..1,color=red),
      plot3d([calculaPuntoSpline(subMallas[2],u,v)[1],
        calculaPuntoSpline(subMallas[2],u,v)[2],
        calculaPuntoSpline(subMallas[2],u,v)[3]],
        'u'=0..1,'v'=0..1,color=blue),
      plot3d([calculaPuntoSpline(subMallas[3],u,v)[1],
        calculaPuntoSpline(subMallas[3],u,v)[2],
        calculaPuntoSpline(subMallas[3],u,v)[3]],
        'u'=0..1,'v'=0..1,color=green),
      plot3d([calculaPuntoSpline(subMallas[4],u,v)[1],
        calculaPuntoSpline(subMallas[4],u,v)[2],
        calculaPuntoSpline(subMallas[4],u,v)[3]],
        'u'=0..1,'v'=0..1,color=khaki),
      dibujaPoligonoControl(malla,
        COLOUR(RGB,0.75,0.75,0.75)),
      dibujaPoligonoControl(subMallas[1],
       COLOUR(RGB, 0.75, 0, 0)),
      dibujaPoligonoControl(subMallas[2],
       COLOUR(RGB, 0, 0.75, 0)),
      dibujaPoligonoControl(subMallas[3],
       COLOUR(RGB, 0, 0, 0.75)),
      dibujaPoligonoControl(subMallas[4],
       COLOUR(RGB, 0.75, 0.75, 0))
   );
end proc:
pintaHijos(MallaDeControl);
pintaHijos(MallaDeControl2);
pintaHijos(MallaDeControl3);
```

A.6. Prueba de la corrección de la subdivisión

```
polinomioSpline := proc(malla)
option remember;
local i, j, B1, B2, polinomio;
   polinomio:=[0,0,0];
   for i from 1 to 4 do
      for j from 1 to 4 do
         unassign('t');
         B1 := subs({'t'='u'}, Bernstein(3,i-1));
```

```
            B2 := subs({'t'='v'}, Bernstein(3,j-1));
            polinomio:= polinomio + malla[i][j] * B1 * B2;
        od;
    od;
return polinomio;
end proc:

ComparaPolinomios := proc(malla)
option remember;
local subMallas, pOrig, p1, p2, p3, p4;
    subMallas := subdivideMalla(malla);
    pOrig := polinomioSpline(malla);
    p1 := polinomioSpline(subMallas[1]);
    p2 := polinomioSpline(subMallas[2]);
    p3 := polinomioSpline(subMallas[3]);
    p4 := polinomioSpline(subMallas[4]);

    #Realizamos las substituciones:

    #Polinomio de la malla original (no substituimos,
    se hace sólo por evitar problemas con el redondeo.
    pOrig := collect(expand(subs({u=1.0*u, v=1.0*v},
                    pOrig)),[u,v]);#, 'distributed');
    print("pOrig: ", pOrig);

    #Parte de abajo a la izquierda
    p1 := collect(expand(subs({u=2.*u, v=2.*v},
                p1)), [u,v]);
    print("p1: ", p1);

    #Parte de arriba a la izquierda
    p2 := collect(expand(subs({u=2*u, v=2*v-1},
                p2)), [u,v]);
    print("p2: ", p2);

    #Parte de abajo a la derecha
    p3 := collect(expand(subs({u=2*u-1, v=2*v},
                p3)), [u,v]);
    print("p3: ", p3);

    #Parte de arriba a la derecha
```

```
    p4 := collect(expand(subs({u=2*u-1, v=2*v-1},
                  p4)), [u,v]);
    print("p4: ", p4);

end proc:
ComparaPolinomios(MallaDeControl);
ComparaPolinomios(MallaDeControl2);
```

A.7. Algoritmo Minmax

A.7.1. Caja contenedora de una malla

Cálculo de la caja

```
calculaCaja := proc (malla)
option remember;
local maxX,minX,maxY,minY,maxZ,minZ,i,j;
minX:=min();
minY:=min();
minZ:=min();
maxX:=max();
maxY:=max();
maxZ:=max();
for i from 1 to 4 do
    for j from 1 to 4 do
        minX:= min(minX, malla[i][j][1]);
        minY:= min(minY, malla[i][j][2]);
        minZ:= min(minZ, malla[i][j][3]);
        maxX:= max(maxX, malla[i][j][1]);
        maxY:= max(maxY, malla[i][j][2]);
        maxZ:= max(maxZ, malla[i][j][3]);
    od;
od;
return [[minX,minY,minZ],[maxX,maxY,maxZ]];
end proc:
cajita:=calculaCaja(MallaDeControl);
```

Dibujo de la caja

```
dibujaCaja := proc (caja)
option remember;
local col, miColor;
```

```
col := rand(255);
miColor := COLOR(RGB, col()/255, col()/255, col()/255);

return pointplot3d([[caja[1][1],caja[1][2],caja[1][3]],
                    [caja[1][1],caja[2][2],caja[1][3]],
                    [caja[2][1],caja[2][2],caja[1][3]],
                    [caja[2][1],caja[1][2],caja[1][3]],
                   [caja[1][1],caja[1][2],caja[1][3]]],
              color=miColor, style=line, thickness=1 ),
       pointplot3d( [[caja[1][1],caja[1][2],caja[2][3]],
                    [caja[1][1],caja[2][2],caja[2][3]],
                    [caja[2][1],caja[2][2],caja[2][3]],
                    [caja[2][1],caja[1][2],caja[2][3]],
                   [caja[1][1],caja[1][2],caja[2][3]]],
              color=miColor, style=line, thickness=1 ),
        pointplot3d( [
                    [caja[1][1],caja[1][2],caja[1][3]],
                  [caja[1][1],caja[1][2],caja[2][3]] ],
              color=miColor, style=line, thickness=1 ),
        pointplot3d( [
                    [caja[1][1],caja[2][2],caja[1][3]],
                  [caja[1][1],caja[2][2],caja[2][3]] ],
              color=miColor, style=line, thickness=1 ),
       pointplot3d( [[caja[2][1],caja[2][2],caja[1][3]],
                  [caja[2][1],caja[2][2],caja[2][3]] ],
              color=miColor, style=line, thickness=1 ),
       pointplot3d( [[caja[2][1],caja[1][2],caja[1][3]],
                  [caja[2][1],caja[1][2],caja[2][3]] ],
               color=miColor, style=line, thickness=1 )

end proc:

display(dibujaCaja(cajita),
dibujaPoligonoControl(MallaDeControl,
                      COLOUR(RGB, 1, 0, 0)));

display(dibujaCaja(calculaCaja(MallaDeControl2)),
dibujaPoligonoControl(MallaDeControl2,
                      COLOUR(RGB, 1, 0, 0)));
```

Cálculo del tamaño de una caja

```
tamCaja:=proc(caja)
option remember;
   return min(
      abs(caja[1][1]-caja[2][1]),
      abs(caja[1][2]-caja[2][2]),
      abs(caja[1][3]-caja[2][3]));
end proc:
tamCaja(cajita);
```

Cálculo del centro de una caja

```
centroCaja:=proc(caja)
option remember;
   return [
      (caja[1][1]+caja[2][1])/2,
      (caja[1][2]+caja[2][2])/2,
      (caja[1][3]+caja[2][3])/2];
end proc:
centroCaja(cajita);
display(
   dibujaCaja(cajita),
   pointplot3d([centroCaja(cajita)], symbol=circle,
               symbolsize=50, color=black),
   dibujaPoligonoControl(MallaDeControl,
                         COLOUR(RGB, 1,0,0)));
```

A.7.2. Recta

Definición de una recta

```
Recta := proc(a,b,c,d,e,f)
   return [a,b,c,d,e,f];
end proc:
```

Dibujo de una recta (adaptada a una caja)

```
dibujaRecta := proc(recta, caja)
option remember;
local valorMin, valorMax,retorno;

if(not(recta[1]=0)) then
```

```
    valorMin := (caja[1][1]-recta[4])/recta[1];
    valorMax := (caja[2][1]-recta[4])/recta[1];
else
    if(not(recta[2]=0)) then
        valorMin := (caja[1][2]-recta[5])/recta[2];
        valorMax := (caja[2][2]-recta[5])/recta[2]
    else
        if(not(recta[3]=0)) then
            valorMin := (caja[1][3]-recta[6])/recta[3];
            valorMax := (caja[2][3]-recta[6])/recta[3]
        else
             valorMin := -2;
             valorMax := 2;
        fi;
    fi;
fi;
retorno := pointplot3d({
    [recta[1]*valorMin+recta[4],
    recta[2]*valorMin+recta[5],
    recta[3]*valorMin+recta[6]],
    [recta[1]*valorMax+recta[4],
    recta[2]*valorMax+recta[5],
    recta[3]*valorMax+recta[6]]},
     connect=true,
    thickness=2);
return retorno;
end proc:
```

Dibujo de una recta dado un parámetro

```
dibujaRectaParametro := proc(recta, parametro)
option remember;
return pointplot3d({
    [recta[1]*parametro+recta[4],
     recta[2]*parametro+recta[5],
     recta[3]*parametro+recta[6]],
    [recta[4], recta[5], recta[6]]},
    connect=true, thickness=1, color=black);
end proc:
```

Generación de un haz de rectas

```
creaHazDeRectas := proc(ancho, alto)
option remember;
   local i, j, retorno, anchoI, altoI;

   retorno := [];
   anchoI := 2/(ancho + 1);
   altoI := 2/(alto + 1);

   for i from 1 to ancho do
      for j from 1 to alto do
         retorno := [op(retorno),
            Recta(evalf(-1+anchoI*i),
            evalf(-1+altoI*j),
            1, 0, 0, -1)];
      od;
   od;
   return retorno;
end proc:
```

Corte de una recta con una caja

```
cortaRectaCaja := proc(recta, caja)
option remember;
local param, valorX, valorY, valorZ, minX, minY,
       minZ, maxX, maxY, maxZ, corte, i;

#Inicialización de la variable corte
corte:=false;

#Extraigo los valores de la caja
minX := caja[1][1];
maxX := caja[2][1];
minY := caja[1][2];
maxY := caja[2][2];
minZ := caja[1][3];
maxZ := caja[2][3];

#Comprobamos que la x tenga parámetro t
if(not(recta[1]=0)) then
#Despejamos para la x minima y la x maxima
for i from 1 to 2 do
```

```
    param:= (caja[i][1] - recta[4])/recta[1];

    #Comprobamos para ambos valores del parámetro
    valorY := recta[2]*param + recta[5];
    valorZ := recta[3]*param + recta[6];

    corte := evalb(
        (valorY >= minY) and
        (valorY <= maxY) and
        (valorZ >= minZ) and
        (valorZ <= maxZ));
    if (corte) then
        break;
    fi;
od;
fi;
if (corte) then
    return true;
else
    #Comprobamos que la y tenga parámetro t
    if(not(recta[2]=0)) then
    #Repetimos con la Y.
    for i from 1 to 2 do
        param:= (caja[i][2] - recta[5])/recta[2];

        #Comprobamos para ambos valores del parámetro
        valorX := recta[1]*param + recta[4];
        valorZ := recta[3]*param + recta[6];

        corte := evalb(
            (valorX >= minX) and
            (valorX <= maxX) and
            (valorZ >= minZ) and
            (valorZ <= maxZ));
        if (corte) then
            break;
        fi;
    od;
    fi;
    if (corte) then
        return true;
```

```
    else
        #Comprobamos que la z tenga parÃ¡metro t
        if(not(recta[3] = 0)) then
        #Finalmente, repetimos con la Z.
        for i from 1 to 2 do
            param:= (caja[i][3] - recta[6])/recta[3];

            #Comprobamos para ambos valores del parametro
            valorX := recta[1]*param + recta[4];
            valorY := recta[2]*param + recta[5];

            corte := evalb(
                (valorX >= minX) and
                (valorX <= maxX) and
                (valorY >= minY) and
                (valorY <= maxY));
            if (corte) then
                break;
            fi;
        od;
        fi;
        if (corte) then
            return true;
        fi;
    fi;
fi;

return false;

end proc:
rec := Recta(3,3,2,0,2,0):
#rec := Recta(3,3,0,0,2,1.2):
display(
    dibujaRecta(rec, cajita),
    dibujaCaja(cajita),
    pintaSpline(MallaDeControl),
    dibujaPoligonoControl(MallaDeControl,
                          COLOUR(RGB, 1, 0, 0))
);
cortaRectaCaja(rec, cajita);
```

A.7.3. Algoritmo principal

Versión iterativa normal

```
minimax:= proc(mallaControl,recta,precision)
option remember;
local colaMallas,puntosCorte,caja,malla,submallas,i,
      pintaMallas, ps, col, cajas;

colaMallas:=queue[new]();
queue[enqueue](colaMallas,mallaControl);
puntosCorte:=[];
pintaMallas := [];
cajas := [];

while(queue[length](colaMallas)>0) do
   #extraemos la primera malla de la cola
   malla:=queue[dequeue](colaMallas);

   #obtenemos la caja de la malla
   caja:=calculaCaja(malla);

   #comprobamos si corta con la recta
   if(cortaRectaCaja(recta,caja)) then

      pintaMallas := [op(pintaMallas), malla];

      #si corta, comparamos con la precisión
      cajas := [op(cajas), dibujaCaja(caja)];
      if(precision>tamCaja(caja)) then
         #apuntamos el punto de corte
         puntosCorte:=[op(puntosCorte),
                       centroCaja(caja)];
      else
         #subdividimos
         submallas:= subdivideMalla(malla);
         for i from 1 to 4 do
            queue[enqueue](colaMallas,submallas[i]);
         od;
      fi;
   fi;
```

```
od;
col := rand(255);
ps := [];
for i from 1 to nops(pintaMallas) do
   ps := [op(ps), dibujaPoligonoControl(pintaMallas[i],
         COLOR(RGB, col()/255, col()/255, col()/255))];
od;
return [puntosCorte, ps, cajas];
end proc:
```

Ejemplo de uso

```
resultado := minimax(MallaDeControl, rec,0.001):
print("Puntos de corte", resultado[1]);
display(
   resultado[2],
   dibujaRecta(rec, cajita),
   pointplot3d(resultado[1])
);
display(
   resultado[3],
   dibujaRecta(rec, cajita),
   pointplot3d(resultado[1])
);
```

Versión recursiva

```
miniMaxR := proc(malla, recta, precision)
 local datos, i, datosNuevos, normal;
 option remember;

   datos := miniMaxRec(malla, recta, precision,
                       0, 1, 0, 1);
   datosNuevos := [];
   for i from 1 to nops(datos) do:
      normal := normalMallaConTresPuntos(datos[i][6]);
      datosNuevos := [op(datosNuevos), [op(datos[i]),
                      normal]];
   od;

   return datosNuevos;
end proc:
```

```
miniMaxRec := proc(malla, recta, precision, uMin, uMax,
                   vMin, vMax)
option remember;
local caja, subMallas, uMedia, vMedia, retorno;

 # Obtenemos la caja de la malla
 caja := calculaCaja(malla);
 # Comprobamos si corta con la recta
   if (cortaRectaCaja(recta,caja)) then
      #Miramos el tamaño de la caja
      if (tamCaja(caja) < precision) then
         #Entonces hemos determinado el punto de corte
         retorno := [[centroCaja(caja), uMin, uMax,
                      vMin, vMax, malla]];
      else
         # Como la caja no es suficientemente pequeña,
         # dividimos.
         subMallas := subdivideMalla(malla);
         # Calculamos las u's y las v's de cada trozo
         uMedia := (uMax+uMin)/2;
         vMedia := (vMax+vMin)/2;
         # Calculamos el punto de corte para cada una
         # de las mallas.
         retorno :=
            [miniMaxRec(subMallas[1], recta, precision,
                        uMin, uMedia, vMin, vMedia)[],
             miniMaxRec(subMallas[2], recta, precision,
                        uMin, uMedia, vMedia, vMax)[],
             miniMaxRec(subMallas[3], recta, precision,
                        uMedia, uMax, vMin, vMedia)[],
             miniMaxRec(subMallas[4], recta, precision,
                        uMedia, uMax, vMedia, vMax)[]]
      fi;
   else
      #La recta no corta a la caja
      retorno := [];
   fi;
return retorno;
end proc:
```

```
resultadoMinimax:=miniMaxR(MallaDeControl, rec, 0.001):
```

A.8. Operaciones con vectores

A.8.1. Cálculo vector con dos puntos

```
vectorDosPuntos:=proc(p1,p2)
local v;
v:=[[],[],[]];
v[1]:=p1[1]-p2[1];
v[2]:=p1[2]-p2[2];
v[3]:=p1[3]-p2[3];
return v;
end proc:

vectorDosPuntos([1,2,3],[3,4,5]);
```

A.8.2. Producto vectorial

```
productoVectorial:=proc(v1,v2)
local v;
    v:=[0,0,0];
    v[1]:=v1[2]*v2[3]-v1[3]*v2[2];
    v[2]:=v1[3]*v2[1]-v1[1]*v2[3];
    v[3]:=v1[1]*v2[2]-v1[2]*v2[1];
return v;
end proc:
```

A.8.3. Producto escalar

```
productoEscalar:=proc(v1,v2)
local salida;
    salida:=v1[1]*v2[1]+v1[2]*v2[2]+v1[3]*v2[3];
return salida;
end proc:

productoEscalar([0,1,1],[1,0,0]);
productoEscalar([1,1,1],[1,0,0]);
```

A.8.4. Módulo de un vector

```
modulo:=proc(v)
```

```
local m;
m:=sqrt((v[1])^2+(v[2])^2+(v[3])^2);
return m;
end proc:

modulo([0,1,1]);
modulo([2,3,6]);
```

A.8.5. Coseno del ángulo entre dos vectores

```
cosenoVectores:=proc(v1,v2)
local coseno,m;
coseno:=v1[1]*v2[1]+v1[2]*v2[2]+v1[3]*v2[3];
m:=modulo(v1)*modulo(v2);
coseno:=coseno/m;
return coseno;
end proc:

cosenoVectores([0,1,1],[1,0,0]);
```

A.8.6. Cálculo del ángulo entre dos vectores

```
anguloVectores:=proc(v1,v2)
local coseno;
coseno:=cosenoVectores(v1,v2);
return arccos(coseno);
end proc:
anguloVectores([0,1,1],[1,0,0]);
```

A.9. Cálculo de la normal por métodos aproximativos

A.9.1. Vector normal de un plano dado por tres puntos

Cálculo utilizando un determinante

```
normalPlano:= proc (a,b,c)
   local m;
   m:= Matrix([['i','j','k'],[b[1]-a[1],b[2]-a[2],
         b[3]-a[3]],[c[1]-a[1],c[2]-a[2],c[3]-a[3]]]);
   return Determinant(m);
end proc:
```

Cálculo manual

```
normalPlanoAmano:=proc(a,b,c)
    local ab,ac,determinante;
    ab:=vectorDosPuntos(b,a);
    ac:=vectorDosPuntos(c,a);
    determinante:= productoVectorial(ab,ac);

    return determinante;
end proc:
```

Ejemplos de uso

```
normalPlano([1,0,0],[0,2,0],[0,0,-1]);
normalPlanoAmano([1,0,0],[0,2,0],[0,0,-1]);
```

A.9.2. Vector normal de una malla a partir de tres puntos

```
normalMallaConTresPuntos:=proc (malla)
    local normal, modu;
    normal := normalPlanoAmano(malla[1][1],
                               malla[1][4],
                               malla[4][4]);
    modu:= modulo(normal);
    return (normal / modu);
end proc:

normalMallaConTresPuntos(MallaDeControl);
```

A.9.3. Vector normal como suma de dos normales

```
normalMallaConDosTriangulos:= proc (malla)
    local normal1,normal2;
    normal1:=normalPlanoAmano(malla[1][1],malla[1][4],
                              malla[4][4]);
    normal2:=normalPlanoAmano(malla[1][1],malla[4][4],
                              malla[4][1]);
    return normal1+normal2;
end proc:

normalMallaConDosTriangulos(MallaDeControl);
```

A.9.4. Cálculo de las normales de los puntos de corte de una malla con una recta

```
normalesPuntosDeCorte:=proc(mallas)
local normales,i,normal;
normales:=[];
for i from 1 to nops(mallas) do
      normal:=normalMallaConTresPuntos(mallas[i]);
      print("Normal: ",normal);
      normales:=[op(normales),normal];
od;
return normales;
end proc:
```

A.9.5. Pinta normal

```
pintaNormal:=proc(normal,punto,val)
 return pointplot3d({[normal[1]*val+punto[1],
                normal[2]*val+punto[2],
                normal[3]*val+punto[3]],punto},
                connect=true,thickness=2);
end proc:
```

A.10. Normal de una superficie en un punto

A.10.1. Derivadas de la superficie de Bézier

Derivadas usando diff

```
calculaDerivada:=proc(malla)
local deriv,polinomio;
unassign('u');
unassign('v');
deriv:=[[0,0,0],[0,0,0]];
polinomio:=polinomioSpline(malla);
deriv[1]:=diff(polinomio,u);
deriv[2]:=diff(polinomio,v);
return deriv;
end proc:
```

Ejemplo de uso

```
der:=calculaDerivada(MallaDeControl):
subs({u=0.5, v=0.5}, der);
```

Derivada "manual"

```
calculaDerivada2:=proc(malla)
local i, j, B1, B2, polinomio, deriv;

    deriv:=[[0,0,0],[0,0,0]];
    polinomio:=[0,0,0];
    for j from 1 to 4 do
        for i from 1 to 3 do
            unassign('t');
            B1 := subs({'t'='u'}, Bernstein(2,i-1));
            B2 := subs({'t'='v'}, Bernstein(3,j-1));
            deriv[1]:= deriv[1]+(malla[i+1][j]*B1*B2)-
                       (malla[i][j])*B1*B2;
        od;
    od;
    for i from 1 to 4 do
        for j from 1 to 3 do
            unassign('t');
            B1 := subs({'t'='u'}, Bernstein(3,i-1));
            B2 := subs({'t'='v'}, Bernstein(2,j-1));
            deriv[2]:= deriv[2]+(malla[i][j+1]*B1*B2)-
                       (malla[i][j])*B1*B2;
        od;
    od;
    deriv[1]:=3*deriv[1];
    deriv[2]:=3*deriv[2];

return deriv;
end proc:
```

Ejemplo de uso

```
subs({u=0.5, v=0.5}, calculaDerivada2(MallaDeControl));
```

A.10.2. Cálculo de la normal

Cálculo del vector normal a la Spline

```
vectorNormal :=proc(malla, r, s)
```

```
local normal,d,d1,d2,m,i;
normal := [0,0,0];
d:=subs({'u'=r,'v'=s},calculaDerivada2(malla));
d1:=d[2];
d2:=d[1];
normal:=productoVectorial(d1,d2);
m := sqrt(normal[1]^2 + normal[2]^2 + normal[3]^2);
normal := normal/m;
return normal;
end proc:
```

Ejemplo de uso

```
vectorNormal(MallaDeControl, 0, 0);
#Dibujo de la normal en la superficie
display(
pointplot3d({calculaPuntoSpline(MallaDeControl,0.5,0.5)
              -vectorNormal(MallaDeControl,0.5,0.5)/5,
        calculaPuntoSpline(MallaDeControl, 0.5, 0.5)},
           connect=true,thickness=2),
    pintaSpline(MallaDeControl, COLOUR(RGB, 1, 0, 0))
)
;
display(
  pointplot3d({
   calculaPuntoSpline(MallaDeControl3,0.5,0.5)-
                        vectorNormal(MallaDeControl3,
                                     0.5,0.5)/5,
      calculaPuntoSpline(MallaDeControl3, 0.5, 0.5)},
      connect=true,thickness=2),
  pintaSpline(MallaDeControl3, COLOUR(RGB, 1, 0, 0))
);
```

A.11. Buffer Z

```
filtraPuntos := proc(resultado)
option remember;
local datos, i;
datos := resultado[1];

for i from 2 to nops(resultado) do
```

```
    if (datos[1][3] > resultado[i][1][3]) then
       datos := resultado[i];
    fi;
od:
return datos;
end proc:
```

A.12. Cálculo de todos los puntos de corte para un haz de rectas

```
calculaPuntosYnormales := proc(malla, ancho, alto)
option remember;
local rectas, resultados, i, retorno;

#Obtenemos el haz de rectas que suponen
# el área de visión
rectas := creaHazDeRectas(ancho,alto);

#Una a una, las intersecamos con la malla
resultados := [];
for i from 1 to nops(rectas) do
    resultados := [op(resultados),
                   miniMaxR(malla,rectas[i],0.001)];
od;

#Ahora, de cada juego de resultados, nos quedamos
#con el que tiene el punto de corte más cercano.
retorno := [];
for i from 1 to nops(resultados) do
    if (nops(resultados[i]) > 0) then
       retorno := [op(retorno),
                   filtraPuntos(resultados[i])];
    fi;
od;

return retorno;
end proc:
```

Ejemplo de uso

```
dibujosRectas := []:
```

```
rectas := creaHazDeRectas(20,15):
for i from 1 to nops(rectas) do
   dibujosRectas := [op(dibujosRectas),
                     dibujaRectaParametro(rectas[i], 5)]:
od:
display(dibujosRectas, pintaSpline(MallaDeControl3));
res := calculaPuntosYnormales(MallaDeControl3, 20,15):
puntos := []:
dibujaNormales := []:
normales:=[]:
for i from 1 to nops(res) do
   puntos := [op(puntos), res[i][1]];
   dibujaNormales := [op(dibujaNormales),
      pointplot3d({res[i][1],
                   (res[i][7]*100000)+res[i][1]},
                   connect=true, thickness=2)];
   normales:=[op(normales),res[i][7]];
od:
display(pointplot3d(puntos), dibujaNormales);
pointplot3d(puntos);
```

A.13. Representación

A.13.1. Dibujo del mapa de bits

```
mapaDeBits := [
   [[1,0,0],[1,0.5,0],[1,1,0],[0,0,1]],
   [[1,0,0.5],[1,0.5,0.5],[1,1,0.5],[0,1,0]],
   [[1,0,1],[1,0.5,1],[1,1,1],[1,0,0]]
]:
#generamos un mapa de bits aleatorio
mapaDeBits2:=[];
color:= rand(256);
for i from 1 to 15 do
   fila:=[];
   for j from 1 to 20 do
      colorAux:=[sin(i*j), sin(i*j), sin(i*j)];
      fila:= [op(fila),colorAux];
   od;
   mapaDeBits2:=[op(mapaDeBits2),fila];
od:
```

```
dibujos := []:

for j from 1 to nops(mapaDeBits2) do
   for i from 1 to nops(mapaDeBits2[1]) do
      punt := [[i-0.5, -j-0.5], [i-0.5, -j+0.5],
               [i+0.5, -j+0.5], [i+0.5, -j-0.5]];
      dibujos := [op(dibujos), polygonplot(punt,
                 color=COLOR(RGB, mapaDeBits2[j][i][1],
           mapaDeBits2[j][i][2], mapaDeBits2[j][i][3]),
                 style=patchnogrid, axes=none,
                  scaling=constrained)];
    od:
od:

display(dibujos);

dibujos := []:
for j from 1 to nops(mapaDeBits) do
   for i from 1 to nops(mapaDeBits[1]) do
      punt := [[i-0.5, -j-0.5], [i-0.5, -j+0.5],
               [i+0.5, -j+0.5], [i+0.5, -j-0.5]];
      dibujos := [op(dibujos),
                  polygonplot(punt,
                              color=COLOR(RGB,
                              mapaDeBits[j][i][1],
                              mapaDeBits[j][i][2],
                              mapaDeBits[j][i][3]),
                              style=patchnogrid,
                              axes=none,
                              scaling=constrained)];
   od:
od:
display(dibujos);
```

A.13.2. Luz

Representación abstracta de una luz

```
representaLuz:=proc(posicion,ro,intensidad)
local luz;
#[posicion de la luz(x,y,z), color de la luz (r,g,b),
```

```
#intensidad de la luz]
luz:=[posicion,ro,intensidad];
return luz;
end proc:
```

Cálculo de la percepción del observador en un punto

```
calculoLuz1P:=proc(o,p,N,luz)
local SF,pF,op,luzReflejada,porc,f,ro,intensidad;
  f:=luz[1]; # posición de la luz
  ro:=luz[2]; #color de la luz
  intensidad:=luz[3];
  luzReflejada:=[[],[],[]];
  pF:=vectorDosPuntos(f,p);
  op:=vectorDosPuntos(p,o);
  SF:=2*(productoEscalar(pF,N)*N)-pF;
  porc:=cosenoVectores(SF,op);
  luzReflejada[1]:=intensidad*porc*ro[1];
  luzReflejada[2]:=intensidad*porc*ro[2];
  luzReflejada[3]:=intensidad*porc*ro[3];
  return luzReflejada;
end proc:
```

Cálculo de la luz en todos los puntos de corte

```
calculoLuz:=proc(o,puntos,normales,luz)
local lucesReflejadas,i;
lucesReflejadas:=[];
for i from 1 to nops(puntos) do
  lucesReflejadas:=[op(lucesReflejadas),
  calculoLuz1P(o,puntos[i],normales[i],luz)];
od;
return lucesReflejadas;
end proc:
```

Ejemplo de uso

```
luz1:=representaLuz([10,10,10],[1,1,0],1);
lucesR:=calculoLuz([0,0,0],puntos,normales,luz1):
dibujaLuces:=[]:
for i from 1 to nops(puntos) do
```

```
    dibujaLuces := [op(dibujaLuces),
                    pointplot3d(puntos[i], thickness=5,
color=COLOR(RGB,lucesR[i][1],
                lucesR[i][2],lucesR[i][3]))];
od:

display(dibujaLuces);
```

A.14. Comparación de los métodos de obtención de la normal

A.14.1. Generación de una lista de ángulos que difieren la normal derivada de la normal aproximada

```
generaAngulos := proc(malla, n, m)
local res, i, normalesA, normalesD,
      uM, vM, angulos, dif;

# Obtenemos los puntos de corte
res := calculaPuntosYnormales(malla, n, m);

# Obtenemos ahora todas las normales ya calculadas
# por el método aproximado.
normalesA := [];
for i from 1 to nops(res) do
   normalesA := [op(normalesA), res[i][7]];
od;

# Calculamos ahora las normales derivando
normalesD := [];
for i from 1 to nops(res) do
   uM := (res[i][2] + res[i][3])/2;
   vM := (res[i][4] + res[i][5])/2;
   normalesD := [op(normalesD),
                 vectorNormal(malla, uM, vM)];
od;
```

```
# Finalmente calculamos los angulos entre ambas
angulos := [];
for i from 1 to nops(res) do
   dif := sqrt((normalesA[i][1] - normalesD[i][1])^2 +
          (normalesA[i][2] - normalesD[i][2])^2 +
          (normalesA[i][3] - normalesD[i][3])^2)/2;
   angulos := [op(angulos), arcsin(dif)*2];
od;
return angulos;
end proc:
```

A.15. Estudio estadístico

A.15.1. Media aritmética

```
#Cálculo de la media aritmética:
mediaAritmetica:= proc(datos)
  local i,media,n;
  media:=0;
  n:=nops(datos);
  for i from 1 to n do
     media:= media + datos[i];
  od;
  return evalf(media/n);
end proc:

#Ejemplo:
angulos:=[2.2,3.2,4.5,5.1];
mediaAritmetica(angulos);
```

A.15.2. Varianza muestral

```
#Cálculo de la varianza muestral:
varianzaMuestral:=proc(datos)
  local media,n,i,varianza;
  n:=nops(datos);
  media:=mediaAritmetica(datos);
  varianza:=0;
  for i from 1 to n do
    varianza:=varianza + (datos[i] - media) ^ 2;
  od;
```

```
  return evalf(varianza/n);
end proc:

#Ejemplo:
varianzaMuestral(angulos);
```

A.15.3. Desviación típica

```
#Cálculo de la desviación típica:
desviacionTipica:=proc(datos)
  return sqrt(varianzaMuestral(datos));
end proc:

#Ejemplo:
desviacionTipica(angulos);
```

A.15.4. Medidas de dispersión

```
 #Cálculo de tres medidas de dispersión de un conjunto
 #de datos (media, varianza y desviación típica):
estudioEstadistico:=proc (datos)
  local i,media,n,varianza,desviacion;
  # cálculo de la media:
  media:=0;
  n:=nops(datos);
  for i from 1 to n do
      media:= media + datos[i];
  od;
  media:=evalf(media/n);
  #cálculo de la varianza:
  varianza:=0;
  for i from 1 to n do
    varianza:=varianza + (datos[i] - media)^2;
  od;
  varianza:=evalf(varianza/n);
  #cálculo de la desviación:
  desviacion:=sqrt(varianza);
  return[media,varianza,desviacion];
end proc:

#Ejemplo:
estudioEstadistico(angulos);
```

A.15.5. Aplicación a los ángulos de las normales

```
 angulos := generaAngulos(MallaDeControl3, 8,6):
estudio := estudioEstadistico(angulos):
print("Media: ",estudio[1]);
print("Varianza: ", estudio[2]);
print("Desviación típica: ", estudio[3]);
```

Apéndice B

Listado de código C++

El presente apéndice contiene el listado de código en C++ que utilizamos en el proyecto. Todo el código C++ utilizado en el proyecto, tanto para la ejecución en Windows, Linux, como para la ejecución en la Virtex II Pro realizando las operaciones de división por software y por hardware, está contenido en los mismos archivos. Para no tener que modificar el código independientemente de la plataforma, tenemos una serie de macros que nos permiten elegir qué código compilar.

B.1. Macros de configuración

B.1.1. `splines.h`

```
#ifndef SPLINES_CONFIG_H
#define SPLINES_CONFIG_H

//Descomentar para compilar en XPS
// #define SPLINES_HARDWARE

//Descomentar para representar sólo en función de Z
// #define BUFFERZ

//Descomentar para mostrar la iluminación especular
#define ESPECULAR

#ifdef SPLINES_HARDWARE
//Includes para hardware
#include "xparameters.h"
```

```
#include "xcache_l.h"
#include "stdio.h"
#include "xgpio.h"
//Descomentar para ejecutar la division de mallas
//en la FPGA.
#define FPGA
#else
//Includes para software
//Descomentar para DEBUG
#define DEBUG
#include <iostream>
#include <string>
#include <sstream>
#include <cstdlib>
#include <SDL.h>
using namespace std;

//Descomentar para que funcione con float. Comentar
//para punto fijo.
#define PFIJO_TEST

//Descomentar para punto fijo corto. Comentar para largo.
// #define PFIJO_CORTO
#endif //SPLINES_HARDWARE

//Includes globales
#include <math.h>

//Macros varias
/**
 * Macro para calcular un color con saturación.
 */
#define SAT_COLOR(x) ((x) > 255 ? 255 : (x))

/**
 * Macro para eliminar un valor negativo.
 */
#define POSIT(x) ((x) < 0 ? 0 : (x))

/**
 * Macro para intercambiar dos float sólo si el primero
```

```
 * es mayor que el segundo.
 */
#define SWAP(l0, l1) if (l1 < l0) { \
    float swap_float = l0; \
    l0 = l1; \
    l1 = swap_float;\
}

/**
 * Macro para calcular el mínimo de dos valores.
 */
#define MIN(a, b) ((a) < (b) ? (a) : (b))

/**
 * Macro para calcular el máximo de dos valores.
 */
#define MAX(a, b) ((a) > (b) ? (a) : (b))

/**
 * Macro para calcular el valor absoluto de un número.
 */
#define ABS(x) ((x) > 0 ? (x) : -(x))

#endif //SPLINES_CONFIG_H
```

B.2. Programa principal

B.2.1. `splines.cpp`

```
#include "splines.h"
#include "punto.h"
#include "mallacontrol.h"
#include "recta.h"
#include "imagen.h"
#include "luces.h"
#include "bernstein.h"

#ifdef SPLINES_HARDWARE
#include "controlgpio.h"
#else
```

```
void dibujaPixel(SDL_Surface *screen, int x, int y,
                 Pixel pix) {
    //Calculamos el color
    Uint32 pixel = SDL_MapRGB(screen->format, pix.R,
                              pix.G, pix.B);

    //Obtenemos la dirección donde hay que pintar
    Uint8 *p = (Uint8 *)screen->pixels +
               y * screen->pitch +
               x * screen->format->BytesPerPixel;

    //Pintamos
    *(Uint32 *)p = pixel;
}

#endif //SPLINES_HARDWARE

int main(int argc, char *argv[]) {
    #ifdef SPLINES_HARDWARE
    //Activamos la cache
    XCache_EnableICache(0xc0000000);
    XCache_EnableDCache(0xc0000000);

    //Inicializamos los GPIOS
    InicializaGPIO();
    print("GPIO inicializado.\r\n");
    #endif //SPLINES_HARDWARE

    // Creamos los puntos de la malla original.
    Punto ***puntos = new Punto**[4];
    for (int i = 0; i < 4; i++) {
        puntos[i] = new Punto*[4];
    }
    puntos[0][0] = new Punto(0.f, .0f, .0f);
    puntos[0][1] = new Punto(.35f, .1f, .1f);
    puntos[0][2] = new Punto(.65f, .7f, .1f);
    puntos[0][3] = new Punto(1.f, .9f, .0f);
    puntos[1][0] = new Punto(0.f, .3f, .25f);
    puntos[1][1] = new Punto(.35f, .9f, .25f);
    puntos[1][2] = new Punto(.65f, .9f, .25f);
    puntos[1][3] = new Punto(.9f, .3f, .25f);
```

```
puntos[2][0] = new Punto(0.f, .3f, .5f);
puntos[2][1] = new Punto(.35f, .9f, .5f);
puntos[2][2] = new Punto(.65f, .9f, .5f);
puntos[2][3] = new Punto(.9f, .3f, .5f);
puntos[3][0] = new Punto(0.f, .7f, .8f);
puntos[3][1] = new Punto(.35f, .3f, .75f);
puntos[3][2] = new Punto(.65f, .3f, .75f);
puntos[3][3] = new Punto(1.f, .2f, .8f);

//Creamos las luces
Luz **luces = new Luz*[3];

//Obtenemos por entrada estándar ancho y alto.
#ifdef SPLINES_HARDWARE
int ancho = 640;
int alto = 480;
#else
int ancho = 800;
int alto = 600;
int anchoPantalla = 800;
int altoPantalla = 600;
#ifdef DEBUG
ancho = 800;
alto = 600;
#endif //DEBUG
#endif //SPLINES_HARDWARE

//Creamos las luces de la escena
luces[0] = new Luz(Punto(0.f,.9f,.9f),0.9,0.1,0.1, .9);
luces[1] = new Luz(Punto(.5f,0.f,.5f), 0.1, .9, 0.1, .7);
luces[2] = new Luz(Punto(1.f,1.f,1.f),0.1, 0.1, .9, .4);

// Creamos las mallas
MallaControl::inicializa(ancho);
MallaControl **mallas = new MallaControl*[1];
mallas[0] = new MallaControl(puntos);

//Creamos las luces de la escena.
Imagen img = Imagen(ancho, alto, mallas, 1, luces, 3);

//Computamos la escena
```

```
#ifdef FPGA
printf("Computamos la imagen (Via HW)\r\n");
#else
printf("Computamos la imagen (VIA SW)\r\n");
#endif
img.computarImagen();

#ifndef SPLINES_HARDWARE
//Representamos la imagen
printf("Pintamos...\r\n");

#ifndef DEBUG
SDL_Init(SDL_INIT_TIMER | SDL_INIT_VIDEO);
SDL_Surface *screen = SDL_SetVideoMode(anchoPantalla,
                                       altoPantalla,
                                       32,
                                       SDL_HWSURFACE
                                       | SDL_DOUBLEBUF);
if (SDL_MUSTLOCK(screen))
    SDL_LockSurface(screen);
SDL_FillRect(screen, 0, 0);
if (SDL_MUSTLOCK(screen))
    SDL_UnlockSurface(screen);
float x2 = ((float) (img.getAncho())) /
           ((float) anchoPantalla);
float y2 = ((float) (img.getAlto())) /
           ((float) altoPantalla);
for (int x = 0; x < anchoPantalla; x++) {
    for (int y = 0; y < altoPantalla; y++) {
        int mX = (int) (((float)x)*x2);
        int mY = (int) (((float)y)*y2);
        dibujaPixel(screen, x, (altoPantalla-1)-y,
                    img.getPixel(mX, mY));
    }
}
SDL_Flip(screen);


SDL_Event e;
while (SDL_WaitEvent(&e)) {
    if (e.type == SDL_QUIT) {
```

```
            SDL_FreeSurface(screen);
            SDL_Quit();
            break;
        } else if (e.type == SDL_KEYDOWN) {
            for (int x = 0; x < anchoPantalla; x++) {
                for (int y = 0; y < altoPantalla; y++) {
                    int mX = (int) (((float)x)*x2);
                    int mY = (int) (((float)y)*y2);
                    dibujaPixel(screen, x, altoPantalla
                                - 1 - y,
                                img.getPixel(mX, mY));
                }
            }
            SDL_Flip(screen);
        }
    }
    #endif //DEBUG
    #endif //SPLINES_HARDWARE

    //Liberamos los recursos
    Recta::limpia();
    for (int i = 0; i < 3; i++)
        delete luces[i];
    delete[] luces;
    delete mallas[0];
    delete[] mallas;

    #ifdef SPLINES_HARDWARE
    //Desactivamos cache
    XCache_DisableDCache();
    XCache_DisableICache();
    #endif //SPLINES_HARDWARE

    return 0;
}
```

B.3. Clase *Bernstein*

B.3.1. `bernstein.h`

```
#ifndef BERNSTEIN_H
```

```
#define BERNSTEIN_H
#include "splines.h"

#ifndef FPGA //Si se calcula con FPGA, no es necesario.
#include "pfijo.h"

/**
 * Representación de los polinomios de Bernstein
 * evaluados en <code>t = 0.5</code>.
 * @author Mónica Jiménez Antón
 * @author Carlos Piñeiro Cordero
 * @author Cristina Valbuena Lledó
 */
class Bernstein {

private:
    /**
     * Matriz de datos calculados.
     */
    PFijo listado[4][4];

    /**
     * Calcula el valor de Bernstein para un grado y un
     * subíndice en <code>t = 0.5</code>.
     * @param n Grado.
     * @param i Subíndice.
     * @return Valor.
     */
    PFijo calculaBernstein(int n, int i);

    /**
     * Instancia del Bernstein (donde están almacenados
     * los valores)
     */
    static Bernstein instancia;

public:
    /**
     * Constructor por defecto. Construye un Bernstein
     * de grado 3.
     */
```

```
    Bernstein();


    /**
     * Destructor.
     */
    ~Bernstein();

    /**
     * Devuelve el valor de Bernstein dados <i>i</i> y
     * <i>n</i>.
     * @param n Grado.
     * @param i Subíndice.
     * @return Valor en <code>t = 0.5</code>.
     */
    PFijo inline & getBernstein(int n, int i) {
        return this->listado[n][i];
    }

    /**
     * Punto de acceso a la clase en modo Singleton.
     * @return Puntero al objeto Singleton.
     */
    static inline Bernstein* getBernstein() {
        return & instancia;
    }
};

#endif //FPGA

#endif //BERNSTEIN_H
```

B.3.2. bernstein.cpp

```
#include "bernstein.h"

#ifndef FPGA

Bernstein Bernstein::instancia = Bernstein();

//Constructor
```

```
Bernstein::Bernstein() {
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            this->listado[i][j] =
                this->calculaBernstein(i,j);
        }
    }
}

//Destructor
Bernstein::~Bernstein() {
// Nada.
}

//Evalúa un polinomio en t = 0.5
PFijo Bernstein::calculaBernstein(int n, int i) {
    PFijo retorno = a_fijo(0.f);
    if ((n == 0) && (i == 0))
        retorno = a_fijo(1.f);
    else if (!((i > n) || (i < 0)))
        retorno = prod(a_fijo(0.5f),
                       calculaBernstein(n-1, i))
                + prod(a_fijo(0.5f),
                       calculaBernstein(n-1, i-1));
    return retorno;
}

#endif //FPGA
```

B.4. Clase *Caja*

B.4.1. caja.h

```
#ifndef CAJA_H
#define CAJA_H
#include "splines.h"

#include "punto.h"

/**
 * Representación de la caja que contiene una Spline.
```

```
 * @author Mónica Jiménez Antón
 * @author Carlos Piñeiro Cordero
 * @author Cristina Valbuena Lledó
 */
class Caja {
private:
    /**
     * Punto mínimo de la caja.
     */
    Punto min;

    /**
     * Punto máximo de la caja.
     */
    Punto max;

    /**
     * Centro de la caja.
     */
    Punto centro;

    /**
     * Tamaño de la caja.
     */
    PFijo tam;

public:
    /**
     * Constructor de la caja a patir de los puntos de
     * la malla de control.
     * @param p Puntos de control.
     */
    Caja(Punto ***p);

    /**
     * Constructor por defecto.
     */
    Caja();

    /**
     * Destructor.
```

```
     */
    ~Caja();

    #ifndef SPLINES_HARDWARE
    /**
     * Representa la caja en un string legible.
     * @return string con los datos de la caja.
     */
    string toString();
    #endif

    /**
     * Permite acceder al centro de la caja.
     * @return Referencia al centro.
     */
    Punto inline & getCentro() {
        return this->centro;
    }

    /**
     * Permite acceder al mínimo de la caja.
     * @return Referencia al mínimo.
     */
    Punto inline & getMin() {
        return this->min;
    }

    /**
     * Permite acceder al máximo de la caja.
     * @return Referencia al máximo.
     */
    Punto inline & getMax() {
        return this->max;
    }

    /**
     * Permite conocer el tamaño de la caja.
     * @return Referencia al tamaño.
     */
    PFijo inline & getTam() {
        return this->tam;
```

```
    }
};

#endif
```

B.4.2. caja.cpp

```
#include "caja.h"
#include "pfijo.h"

//Constructor
Caja :: Caja() {
    // Nada
}

//Constructor a partir de los puntos.
Caja::Caja(Punto ***p) {
    min.x = p[0][0]->x;
    min.y = p[0][0]->y;
    min.z = p[0][0]->z;
    max.x = min.x;
    max.y = min.y;
    max.z = min.z;

    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            min.x = MIN(min.x, p[i][j]->x);
            max.x = MAX(max.x, p[i][j]->x);
            min.y = MIN(min.y, p[i][j]->y);
            max.y = MAX(max.y, p[i][j]->y);
            min.z = MIN(min.z, p[i][j]->z);
            max.z = MAX(max.z, p[i][j]->z);
        }
    }

    tam = MAX(MAX(max.x - min.x, max.y - min.y),
              max.z - min.z);
#ifndef PFIJO_TEST
    centro.x = suma_fijo(min.x, max.x) >> 1;
    centro.y = suma_fijo(min.y, max.y) >> 1;
    centro.z = suma_fijo(min.z, max.z) >> 1;
```

```
#else
    centro.x = (min.x + max.x) / 2.;
    centro.y = (min.y + max.y) / 2.;
    centro.z = (min.z + max.z) / 2.;
#endif
}

#ifndef SPLINES_HARDWARE
//Representación en string
string Caja::toString( ) {
    stringstream strm;
    strm << "[[" << a_float(min.x) << ", ";
    strm << a_float(min.y) << ", " << a_float(min.z)
    << "], [" << a_float(max.x) << ", "
    << a_float(max.y) << ", " << a_float(max.z)
    << "]]";
    return strm.str();
}
#endif

//Destructor
Caja::~ Caja() {
    // Nada.
}
```

B.5. Funciones para el control de los GPIOs

B.5.1. controlgpio.h

```
#ifndef CONTROLGPIO_H
#define CONTROLGPIO_H
#include "splines.h"

#ifdef SPLINES_HARDWARE //Si no está definido,
//no se compila nada.

//16 bits de salida de control,
//16 de entrada de control
#define MASK_CONTROL  0xFFFF0000
//Señales de control
#define SET           0x00020000
```

```
#define ESCRITO         0x00100000
#define GET             0x00040000
#define LEIDO           0x00080000
#define OK              0x00000001
#define READY           0x00000002

//Ajuste del sentido de los registros tri-estado de los
//GPIOs de datos.
#define INPUT           0xFFFFFFFF
#define OUTPUT          0x00000000

/**
 * Esta función inicializa los GPIOs
 */
void InicializaGPIO();

/**
 * La llamada a esta función no retorna hasta que el
 * flag esté a 1.
 */
void EsperaFlag(unsigned int flag);

//Las siguientes funciones se definen como macros por
//cuestion de eficiencia.
#define Lee() (XGpio_DiscreteRead(&gpio, 1))
#define Escribe(x) (XGpio_DiscreteWrite(&gpio, 1, (x)))
#define Control(x) (XGpio_DiscreteWrite(&gpio, 2, (x)))
#define AjustaLectura() XGpio_SetDataDirection(&gpio, \
        1, INPUT);
#define AjustaEscritura() XGpio_SetDataDirection(&gpio,\
        1, OUTPUT);

//Declaramos la variable para que funcionen las macros
extern XGpio gpio;

#endif //SPLINES_HARDWARE
#endif //CONTROLGPIO_H
```

B.5.2. controlgpio.cpp

```
#include "controlgpio.h"
```

```
#ifdef SPLINES_HARDWARE //Si no está definido,
XGpio gpio; //Instancia del GPIO

//Espera un flag.
void EsperaFlag(unsigned int mask) {
    while (!(XGpio_DiscreteRead(&gpio, 2) & mask))
        ;
}

//Inicializa los GPIOs
void InicializaGPIO() {
    //Inicializamos el GPIO.
    XGpio_Initialize(&gpio, XPAR_GPIO_DEVICE_ID);

    //Ajustamos la dirección del control
    XGpio_SetDataDirection(&gpio, 2, MASK_CONTROL);

    //Ponemos a 0 las señales de control
    XGpio_DiscreteWrite(&gpio, 2, 0x0);
}

#endif //SPLINES_HARDWARE
```

B.6. Clase *Corte*

B.6.1. `corte.h`

```
#ifndef CORTE_H
#define CORTE_H
#include "splines.h"

#include "punto.h"
#include "vec.h"

/**
 * Representación del corte entre una recta y una Spline.
 * @author Mónica Jiménez Antón
 * @author Carlos Piñeiro Cordero
 * @author Cristina Valbuena Lledó
 */
```

```
class Corte {
private:
    /**
     * Punto de corte.
     */
    Punto punto;

    /**
     * Normal a la superficie en el punto de corte.
     */
    Vec normal;

public:

    /**
     * Constructor por defecto
     */
    Corte();

    /**
     * Constructor con parámetros
     * @param p Punto de corte.
     * @param v Normal a la superficie.
     */
    Corte(Punto* p,Vec* v);

    /**
     * Destructor por defecto
     */
    ~Corte();

    #ifndef SPLINES_HARDWARE
    /**
     * Imprime un corte en un formato legible.
     * @return string con el corte impreso.
     */
    string toString();
    #endif

    /**
     * Accesora al punto de corte.
```

```
     * @return Referencia al punto de corte.
     */
    Punto inline & getPunto() {
        return this->punto;
    }

    /**
     * Accesora a la normal al punto de corte.
     * @return Referencia a la normal.
     */
    Vec inline & getNormal() {
        return this->normal;
    }

    /**
     * Permite conocer la Z del corte.
     * @return float con el valor.
     */
    PFijo inline & getZ() {
        return this->punto.z;
    }
};

#endif
```

B.6.2. corte.cpp

```
#include "corte.h"

//Constructor
Corte::Corte() {
    //Nada.
}

//Constructor.
Corte::Corte(Punto* p,Vec* n) {
    this->punto.x = p->x;
    this->punto.y = p->y;
    this->punto.z = p->z;
    this->normal.set(*n);
}
```

```
//Destructor.
Corte::~Corte() {
    //Nada
}

#ifndef SPLINES_HARDWARE
//Convierte a un formato legible.
string Corte:: toString() {
    stringstream strm;
    strm << "Punto de corte: " << punto.toString()
    << "; Normal: " << normal.toString();
    return strm.str();
}
#endif //Sólo compilar si está en modo SW.
```

B.7. Clase *Imagen*

B.7.1. imagen.h

```
#ifndef IMAGEN_H
#define IMAGEN_H
#include "splines.h"

#include "punto.h"
#include "pfijo.h"
#include "recta.h"
#include "luces.h"
#include "mallacontrol.h"
#include "corte.h"

/**
 * Estructura de un pixel.
 */
typedef struct RGB {
    unsigned char R;
    unsigned char G;
    unsigned char B;
}
Pixel;
```

```
/**
 * Representación de una escena que contiene varias
 * mallas de control y
 * varias luces.
 * @author Mónica Jiménez Antón
 * @author Carlos Piñeiro Cordero
 * @author Cristina Valbuena Lledó
 */
class Imagen {
private:
    /**
     * Anchura de la imagen, en píxeles.
     */
    int ancho;

    /**
     * Altura de la imagen, en píxeles.
     */
    int alto;

    /**
     * Mallas de la escena.
     */
    MallaControl** mallas;

    /**
     * Número de mallas de la escena.
     */
    int nMallas;
    #ifndef SPLINES_HARDWARE
    /**
     * Píxeles de la imagen.
     */
    Pixel*** pixs;
    #endif

    /**
     * Luces de la escena.
     */
    Luz **luces;
```

```
    /**
     * Número de luces de la escena.
     */
    int nLuces;

    /**
     * calcula la suma de todas las luces en un punto
     * @param x Coordenada x del pixel.
     * @param y Coordenada y del pixel.
     * @return Vector con los 3 componentes de la luz.
     */
    float* calcularLuces(int x, int y, Corte *c);

    /**
     * Función para el cómputo de cada pixel.
     */
    void computarPixel(int x, int y);

public:

    /**
     * Construye una imagen vacía.
     */
    Imagen();

    /**
     * Construye una imagen dada una lista de mallas y
     * una resolución.
     * @param ancho Anchura de la imagen.
     * @param alto Altura de la imagen.
     * @param mallas Mallas de la escena.
     * @param nMallas Número de mallas de la escena.
     * @param luces Luces de la escena.
     * @param nLuces Número de luces de la escena.
     */
    Imagen(int ancho, int alto, MallaControl **mallas,
           int nMallas, Luz **luces, int nLuces);

    /**
     * Destruye una imagen.
     */
```

```
    ~Imagen();

    #ifndef SPLINES_HARDWARE
    /**
     * Permite acceder a los píxeles de la imagen.
     * @return Devuelve una referencia al pixel.
     */
    Pixel inline & getPixel(int x, int y) {
        return *this->pixs[x][y];
    }

    /**
     * Devuelve el ancho de la imagen.
     * @return Int con el ancho.
     */
    int inline & getAncho() {
        return this->ancho;
    }

    /**
     * Devuelve el alto de la imagen.
     * @return Int con el alto.
     */
    int inline & getAlto() {
        return this->alto;
    }
    #endif

    /**
     * Computa la imagen, transformando los datos en
     * una imagen.
     */
    void computarImagen();
};

#endif
```

B.7.2. imagen.cpp

```
#include "imagen.h"
```

```
#ifdef SPLINES_HARDWARE
void dibujaPixel(int x, int y, Pixel & pix) {
    //Puntero a la pantalla
    volatile static unsigned int *pantalla =
        (volatile unsigned int*) 0x07E00000;

    //Color
    volatile unsigned int color = 0x00 |
                            (pix.R << 16) | (pix.G << 8) | pix.B;

    //Pintamos
    pantalla[x + 1024 * (480 - y)] = color;
}
#endif

//Destructor
Imagen::~Imagen() {
    #ifndef SPLINES_HARDWARE
    for (int i = 0; i < ancho; i++) {
        for (int j = 0; j < alto; j++)
            delete this->pixs[i][j];
        delete[] this->pixs[i];
    }
    delete[] this->pixs;
    #endif
}

//Constructor.
Imagen::Imagen() {
    this->ancho = 1;
    this->alto = 1;
    #ifndef SPLINES_HARDWARE
    this->pixs = NULL;
    #endif
    this->luces = NULL;
}

//Constructor
Imagen::Imagen(int ancho, int alto, MallaControl **mallas,
               int nMallas, Luz **luces, int nLuces) {
    //Copiamos los datos de entrada.
```

```
    this->ancho = ancho;
    this->alto = alto;
    this->mallas = mallas;
    this->nMallas = nMallas;
    this->nLuces = nLuces;

    //Creamos el haz de rectas.
    Recta::creaHazRectas(ancho, alto);

    //Reservamos espacio para la imagen.
    #ifndef SPLINES_HARDWARE
    this->pixs = new Pixel**[ancho];
    for (int i = 0; i < ancho; i++) {
        this->pixs[i] = new Pixel*[alto];
        for (int j = 0; j < alto; j++) {
            this->pixs[i][j] = new Pixel;
        }
    }
    #else
    //Limpiamos la pantalla
    Pixel pix;
    pix.R = 0xFF;
    pix.G = 0xFF;
    pix.B = 0xFF;
    for (int x = 0; x < 640; x++)
        for (int y = 0; y < 480; y++)
            dibujaPixel(x, y, pix);
    #endif
    this->luces = luces;
}

//Calcula la imagen.
void Imagen :: computarImagen() {
    //Calculamos uno a uno todos los puntos de corte.
    for (int x = 0; x < ancho; x++) {
        for (int y = 0; y < alto; y++) {
            this->computarPixel(x,y);
        }
        #ifdef SPLINES_HARDWARE
        print(".");
        #else
```

```
        cout << ((float) x) / ((float) ancho) * 100
        << '%' << endl;
        #endif
    }
}

//Computa un pixel.
void Imagen :: computarPixel(int x, int y) {
    //Extraemos la recta correspondiente a este pixel.
    Recta *r = Recta::getHaz()[x][y];

    //Computamos todos los puntos de corte.
    Corte *corte = mallas[0]->corte(*r);
    for (int i = 1; i < nMallas; i++) {
        Corte *c = mallas[i]->corte(*r);
        if (corte->getPunto().z == a_fijo(1.f)) {
            delete corte;
            corte = c;
        } else if (c->getPunto().z == a_fijo(1.f)) {
            delete c;
        } else if (c->getPunto().z < corte->getPunto().z) {
            delete corte;
            corte = c;
        } else {
            delete c;
        }
    }

    //Obtenemos el color del pixel para este punto.
    float colorR, colorG, colorB;
    colorR = colorG = colorB = 0.f;
    if (corte->getZ() != a_fijo(1.f)) {
        #ifdef ZBUFFER
        colorR = colorG = colorB = 255.f *
                                   a_float(corte->getZ());
        #else
        float* colores = calcularLuces(x, y, corte);
        colorR = SAT_COLOR(colores[0] * 255.f);
        colorG = SAT_COLOR(colores[1] * 255.f);
        colorB = SAT_COLOR(colores[2] * 255.f);
        delete[] colores;
```

```
        #endif //ZBUFFER
    }
    #ifndef SPLINES_HARDWARE
    this->pixs[x][y]->R = (unsigned char) colorR;
    this->pixs[x][y]->G = (unsigned char) colorG;
    this->pixs[x][y]->B = (unsigned char) colorB;
    #else
    Pixel pix;
    pix.R = (unsigned char) colorR;
    pix.G = (unsigned char) colorG;
    pix.B = (unsigned char) colorB;
    dibujaPixel(x, y, pix);
    #endif //SPLINES_HARDWARE
    delete corte;
}

//Calcula la iluminación en un pixel.
float* Imagen::calcularLuces(int x, int y, Corte *c) {
    float *colores = new float[3];
    colores[0] = 0.f;
    colores[1] = 0.f;
    colores[2] = 0.f;

    for (int i = 0; i < nLuces; i++) {
        float p = luces[i]->calculoDifusa(c->getPunto(),
                                          c->getNormal());
        #ifdef ESPECULAR
        float especular = luces[i]->calculoEspecular(
                              c->getPunto(), c->getNormal());
        especular = POSIT(50*especular -49.5);
        colores[0] += luces[i]->calculoLuzR(p) + especular;
        colores[1] += luces[i]->calculoLuzG(p) + especular;
        colores[2] += luces[i]->calculoLuzB(p) + especular;
        #else
        colores[0] += luces[i]->calculoLuzR(p);
        colores[1] += luces[i]->calculoLuzG(p);
        colores[2] += luces[i]->calculoLuzB(p);
        #endif //ESPECULAR

    }
    return colores;
```

```
}
```

B.8. Clase *Luz*

B.8.1. luces.h

```
#ifndef LUZ_H
#define LUZ_H
#include "splines.h"

#include "vec.h"
#include "punto.h"
#include "pfijo.h"

/**
 * Representación de una luz.
 * @author Mónica Jiménez Antón
 * @author Carlos Piñeiro Cordero
 * @author Cristina Valbuena Lledó
 */
class Luz {
private:
    /**
     * Posición de la luz.
     */
    Punto posicion;

    /**
     * Valor en rojo de la luz.
     */
    float r;

    /**
     * Valor en verde de la luz.
     */
    float g;

    /**
     * Valor en azul de la luz.
     */
    float b;
```

```
    /**
     * Intensidad de la luz.
     */
    float intensidad;

    /**
     * Coordenada X del punto del observador.
     */
    static float observadorX;

    /**
     * Coordenada Y del punto del observador.
     */
    static float observadorY;

    /**
     * Coordenada Z del punto del observador.
     */
    static float observadorZ;

public:
    /**
     * Constructor por defecto
     */
    Luz();

    /**
     * Constructor paramétrico.
     * @param pos Posición de la luz.
     * @param r Valor de rojo.
     * @param g Valor de verde.
     * @param b Valor de azul.
     * @param intensidad Intensidad de la luz.
     */
    Luz(Punto pos, float r, float g, float b,
        float intensidad);

    /**
     *  Destructor por defecto
     */
```

```
    ~Luz();

    /**
     * Permite acceder a la posición que ocupa la luz.
     * @return Referencia a la posición de la luz.
     */
    Punto inline & getPos() {
        return posicion;
    }

    /**
     * Cantidad de luz roja que se ve en un punto.
     * @param porc Porcentaje de la luz que se ve.
     * @return Cantidad total de luz roja.
     */
    float inline calculoLuzR(float porc) {
        return porc * intensidad * r;
    }

    /**
     * Cantidad de luz verde que se ve en un punto.
     * @param porc Porcentaje de la luz que se ve.
     * @return Cantidad total de luz verde.
     */
    float inline calculoLuzG(float porc) {
        return porc * intensidad * g;
    }

    /**
     * Cantidad de luz azul que se ve en un punto.
     * @param porc Porcentaje de la luz que se ve.
     * @return Cantidad total de luz azul.
     */
    float inline calculoLuzB(float porc) {
        return porc * intensidad * b;
    }

    /**
     * Porcentaje de luz difusa que refleja una
     * superficie en un punto.
     * @param p Punto de la superficie.
```

```
     * @param N Normal a la superficie.
     * @return Cantidad en tanto por uno.
     */
    float calculoDifusa(Punto &p,Vec &N);

    #ifdef ESPECULAR
    /**
     * Porcentaje de luz especular que refleja una
     * superficie en un punto.
     * @param p Punto de la superficie.
     * @param N Normal a la superficie.
     * @return Cantidad de luz especular.
     */
    float calculoEspecular(Punto &p, Vec &N);
    #endif

};
#endif
```

B.8.2. luces.cpp

```
#include "luces.h"

float Luz::observadorX = 0.5f;
float Luz::observadorY = 0.5f;
float Luz::observadorZ = 2.f;

//Constructor
Luz::Luz() {
    //Nada
}

//Constructor
Luz::Luz(Punto pos, float r, float g, float b,
         float intensidad) {
    this->posicion = pos;
    this->r = r;
    this->g = g;
    this->b = b;
    this->intensidad = intensidad;
}
```

```
//Cálculo del pocentaje difuso reflejado.
float Luz::calculoDifusa(Punto &p,Vec &N) {
    //Calculamos el vector Punto-FocoLuz
    Vec pf(a_float(posicion.x) - a_float(p.x),
           a_float(posicion.y) - a_float(p.y),
           a_float(posicion.z) - a_float(p.z));

    //Hacemos que el vector normal mire hacia la luz.
    Vec normal(N);
    if (pf.escalar(normal) < 0)
        normal.invertir();

    //Calculamos el vector Punto-Observador
    Vec po(observadorX - a_float(p.x),
           observadorY - a_float(p.y),
           observadorZ - a_float(p.z));

    //Si la normal y el vector anterior forman un ángulo
    //mayor que 90, entonces el observador no puede ver
    //esta luz.
    if (po.escalar(normal) < 0)
        return 0;

    //Normalizamos el vector Punto-Foco
    pf.norm();

    //Devolvemos la cantidad de luz difusa según el
    //modelo de Phong.
    return normal.escalar(pf);
}
//Destructor
Luz::~Luz() {
    //Nada.
}

#ifdef ESPECULAR //Si no está definido, no hay que compilar.
float Luz::calculoEspecular(Punto & p, Vec & N) {
    //Calculamos el vector Punto-FocoLuz
    Vec pf(a_float(posicion.x) - a_float(p.x),
           a_float(posicion.y) - a_float(p.y),
```

```
            a_float(posicion.z) - a_float(p.z));

    //Hacemos que el vector normal mire hacia la luz.
    Vec normal(N);
    if (pf.escalar(normal) < 0)
        normal.invertir();

    //Calculamos el vector Punto-Observador
    Vec po(observadorX - a_float(p.x),
           observadorY - a_float(p.y),
           observadorZ - a_float(p.z));

    //Si la normal y el vector anterior forman un ángulo
    //mayor que 90, entonces el observador no puede ver
    //esta luz.
    if (po.escalar(normal) < 0)
        return 0;

    //Calculamos el vector reflejado:
    //R = (2(V.escalar(N))*N - V
    Vec R(normal);
    R.multModulo(2.f * pf.escalar(normal));
    R.resta(pf);
    po.norm();
    R.norm();

    //Devolvemos la cantidad de luz especular según
    //el modelo de Phong
    return R.escalar(po);
}
#endif
```

B.9. Clase *MallaControl*

B.9.1. `mallacontrol.h`

```
#ifndef MALLACONTROL_H
#define MALLACONTROL_H
#include "splines.h"

#include "punto.h"
```

```
#include "pfijo.h"
#include "caja.h"
#include "recta.h"
#include "corte.h"
#include "vec.h"

/**
 * Representación de una Malla de control.
 * @author Mónica Jiménez Antón
 * @author Carlos Piñeiro Cordero
 * @author Cristina Valbuena Lledó
 */
class MallaControl {
private:
    /**
     * Array de puntos.
     */
    Punto ***puntos;

    /**
     * Array de submallas.
     */
    MallaControl **subMallas;

    /**
     * Caja contenedora de la malla de control.
     */
    Caja c;

    /**
     * Anchura máxima
     */
    static float anchura;

    /**
     * Divide la malla en 4.
     */
    void dividir();

    /**
     * Calcula el punto de corte con una recta.
```

```
     * @param r Recta a intersecar.
     * @return punto de corte o NULL si no se corta.
     */
    Corte* calculaCorte(Recta &r);

    /**
     * Destruye los puntos de la malla.
     */
    void destruyePuntos();

    /**
     * Destruye las submallas.
     */
    void destruyeSubmallas();

    #ifndef FPGA
    /**
     * Calcula un nodo intermedio.
     * @param i Índice i.
     * @param j Índice j.
     * @param r Índice r.
     * @param s Índice s.
     * @return Puntero al punto calculado.
     */
    Punto* nodoIntermedio(int i, int j, int r, int s);
    #endif

    /**
     * Devuelve el corte más cercano y destruye los
     * demás.
     * @param c1 Corte a considerar.
     * @param c2 Corte a considerar.
     * @param c3 Corte a considerar.
     * @param c4 Corte a considerar.
     * @return Corte más cercano o NULL si son todos
     * NULL.
     */
    Corte* corteCercano(Corte* c1, Corte* c2,
                        Corte* c3, Corte* c4);
```

```
    /**
     * Vector normal a la superficie.
     */
    Vec *normal;

public:
    /**
     * Constructor.
     */
    MallaControl();

    /**
     * Constructor.
     * @param puntos Matriz de nodos de la malla de
     * control.
     */
    MallaControl(Punto ***puntos);

    /**
     * Destructor.
     */
    ~MallaControl();

    #ifndef SPLINES_HARDWARE
    /**
     * Representación legible de la malla.
     * @return string con los datos formateados.
     */
    string toString();
    #endif

    /**
     * Calcula un punto de corte.
     * @param r Recta con la que corta.
     * @return Nunca devuelve NULL. Si no corta,
     * devuelve el punot (0, 0, -2).
     */
    Corte* corte(Recta &r);

    /**
     * Inicializa la clase, preparándola para imágenes
```

```
     * con "ancho" píxeles.
     * @param ancho Anchura de la imagen, en píxeles.
     */
    static void inicializa(int ancho);

    #ifdef FPGA //Compila si estamos en modo hardware.
    /**
     * Computa los puntos de las submallas vía hardware.
     * @param subPuntos Almacenaje para los resultados.
     */
    void computaPuntos(Punto**** subPuntos);
    #endif //FPGA
};

#endif
```

B.9.2. mallacontrol.cpp

```
#include "mallacontrol.h"
#include "recta.h"
#include "caja.h"
#include "vec.h"
#include "bernstein.h"
#include "corte.h"
#ifdef SPLINES_HARDWARE
#include "controlgpio.h"
#endif

//Anchura de la pantalla donde se muestra la malla.
float MallaControl :: anchura;

//Constructor.
MallaControl :: MallaControl() {
    this->puntos = NULL;
    this->subMallas = NULL;
    this->normal = NULL;
}

//Destructor de los puntos.
void MallaControl :: destruyePuntos() {
    if (this->puntos != NULL) {
```

```
        for (int i = 0; i < 4; i++) {
            for (int j = 0; j < 4; j++)
                delete this->puntos[i][j];
            delete[] this->puntos[i];
        }
        delete[] this->puntos;
        this->puntos = NULL;
    }
}

//Destructor de las submallas.
void MallaControl :: destruyeSubmallas() {
    if (this->subMallas != NULL) {
        for (int i = 0; i < 4; i++)
            delete this->subMallas[i];
        delete[] this->subMallas;
        this->subMallas = NULL;
    }
}

//Destructor.
MallaControl::~MallaControl() {
    this->destruyePuntos();
    this->destruyeSubmallas();
    if (this->normal != NULL)
        delete this->normal;
}

//Constructor.
MallaControl :: MallaControl(Punto ***puntos) {
    this->puntos = puntos;
    this->c = Caja(puntos);
    this->subMallas = NULL;
    this->normal = NULL;
}

#ifndef FPGA
//Cálculo de un nodo intermedio.
Punto* MallaControl::nodoIntermedio(int i, int j, int r,
                                    int s) {
    Bernstein *b = Bernstein::getBernstein();
```

```
    Punto *retorno = new Punto(a_fijo(0.f), a_fijo(0.f),
                               a_fijo(0.f));
    for (int k = 0; k <= r; k++) {
        for (int l = 0; l <= s; l++) {
            PFijo B = prod(b->getBernstein(r, k),
                           b->getBernstein(s, l));
            #ifdef PFIJO_CORTO
            retorno->x = suma_fijo(retorno->x,
                                   prod(puntos[i+k][j+l]->x,
                                        B));
            retorno->y = suma_fijo(retorno->y,
                                   prod(puntos[i+k][j+l]->y,
                                        B));
            retorno->z = suma_fijo(retorno->z,
                                   prod(puntos[i+k][j+l]->z,
                                        B));
            #else
            #ifdef PFIJO_TEST
            retorno->x += puntos[i+k][j+l]->x * B;
            retorno->y += puntos[i+k][j+l]->y * B;
            retorno->z += puntos[i+k][j+l]->z * B;
            #else
            retorno->x += prod(this->puntos[i+k][j+l]->x,
                               B);
            retorno->y += prod(this->puntos[i+k][j+l]->y,
                               B);
            retorno->z += prod(this->puntos[i+k][j+l]->z,
                               B);
            #endif //PFIJO_TEST
            #endif //PFIJO_CORTO
        }
    }
    return retorno;
}
#endif

#ifndef SPLINES_HARDWARE
//Formateo de la malla en un string.
string MallaControl :: toString( ) {
    stringstream strm;
    for (int i = 0; i < 4; i++)
```

```
        for (int j = 0; j < 4; j++)
            strm << "punto[" << i << "][" << j << "] = "
            << this->puntos[i][j]->toString() << endl;
    return strm.str();
}

#endif //SPLINES_HADWARE

//División de una malla.
void MallaControl::dividir() {
    // Creamos los 4 arrays de puntos.
    Punto ****subPuntos = new Punto***[4];
    for (int k = 0; k < 4; k++) {
        subPuntos[k] = new Punto**[4];
        for (int i = 0; i < 4; i++) {
            subPuntos[k][i] = new Punto*[4];
            #ifdef FPGA
            for (int j = 0; j < 4; j++)
                subPuntos[k][i][j] = new Punto();
            #endif //FPGA
        }
    }

    #ifdef FPGA
    //Calculamos las submallas vía hardware.
    computaPuntos(subPuntos);
    #else
    //Calculamos las submallas vía software.
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            subPuntos[0][i][j] =
                this->nodoIntermedio(0,0,j,i);
            subPuntos[1][i][j] =
                this->nodoIntermedio(0,i,j,3-i);
            subPuntos[2][i][j] =
                this->nodoIntermedio(j,0,3-j,i);
            subPuntos[3][i][j] =
                this->nodoIntermedio(j,i,3-j,3-i);
        }
    }
    #endif //FPGA
```

```
    //Creamos las 4 mallas de control
    this->subMallas = new MallaControl*[4];
    for (int i = 0; i < 4; i++)
        this->subMallas[i] = new MallaControl(subPuntos[i]);

    //Eliminamos el array que habíamos creado por comodidad
    delete[] subPuntos;

    //Como ya nunca harán falta, eliminamos los puntos
    //de la malla original.
    this->destruyePuntos();
}

//Cálculo de un corte.
Corte* MallaControl :: corte(Recta &r) {
    Corte* retorno = this->calculaCorte(r);
    if (retorno == NULL) {
        //Este valor nos permite saber que el punto no es
        //válido.
        Punto *p = new Punto(a_fijo(0.f), a_fijo(0.f),
                             a_fijo(1.f));
        Vec *v = new Vec();
        retorno = new Corte(p, v);
        delete p;
        delete v;
    }
    return retorno;
}

//Cálculo de un corte.
Corte* MallaControl :: calculaCorte(Recta &r) {
    Corte *corte = NULL;

    //Comprobamos si la recta corta a la caja
    if (r.corta(c)) {
        //Comprobamos el tamaño de la caja
        PFijo tamMax = a_fijo(anchura * a_float(
                                  c.getCentro().z) +
                              anchura);
        if (c.getTam() <= tamMax) {
```

```
            //Se calcula la normal
            if (this->normal == NULL)
                this->normal = new Vec(puntos[0][0],
                                       puntos[0][3],
                                       puntos[3][3]);

            //No habrá más divisiones, así que se
            //destruyen los puntos.
            this->destruyePuntos();

            // Entonces podemos tomar el centro de la
            //caja como un punto de corte.
            corte = new Corte(&c.getCentro(), this->normal);
        } else {
            //Dividimos y obtenemos los puntos de corte
            //de los hijos
            if (subMallas == NULL)
                this->dividir();

            //Calculamos los 4 posibles cortes.
            Corte* c1 = subMallas[0]->calculaCorte(r);
            Corte* c2 = subMallas[1]->calculaCorte(r);
            Corte* c3 = subMallas[2]->calculaCorte(r);
            Corte* c4 = subMallas[3]->calculaCorte(r);

            //Nos quedamos con el más cercano.
            corte = this->corteCercano(c1, c2, c3, c4);
        }
    }
    return corte;
}

//Inicializa la clase.
void MallaControl::inicializa(int ancho) {
    anchura = 1.f/((float)(ancho + 1));
}

//Devuelve el corte más cercano.
Corte * MallaControl::corteCercano(Corte * c1, Corte *
                                   c2, Corte * c3,
                                   Corte * c4) {
```

```
    Corte *retorno;
    retorno = c1;
    if (retorno != NULL) {
        if (c2 != NULL) {
            if (retorno->getZ() > c2->getZ()) {
                delete c2;
            } else {
                delete retorno;
                retorno = c2;
            }
        }
    } else {
        retorno = c2;
    }
    if (retorno != NULL) {
        if (c3 != NULL) {
            if (retorno->getZ() > c3->getZ()) {
                delete c3;
            } else {
                delete retorno;
                retorno = c3;
            }
        }
    } else {
        retorno = c3;
    }
    if (retorno != NULL) {
        if (c4 != NULL) {
            if (retorno->getZ() > c4->getZ()) {
                delete c4;
            } else {
                delete retorno;
                retorno = c4;
            }
        }
    } else {
        retorno = c4;
    }
    return retorno;
}
```

```
#ifdef FPGA
//Computa los subPuntos en el hardware especializado
void MallaControl :: computaPuntos(Punto**** subPuntos) {

    //Procesamos la coordenada X
    //Primero ajustamos el canal de salida
    AjustaEscritura();

    //Uno a uno volcamos los puntos
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            //Fijamos los datos
            Escribe(puntos[i][j]->x);

            //Ajustamos las señales
            Control(SET);

            //Esperamos a que haya grabado
            EsperaFlag(OK);

            //Ajustamos las señales de control
            Control(ESCRITO);
        }
    }

    //Negamos todo
    Control(0x0);

    //Nos preparamos para leer
    AjustaLectura();

    //Uno a uno leemos todos los datos
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            for (int k = 0; k < 4; k++) {
                //Activamos GET
                Control(GET);

                //Esperamos a que nos diga OK
                EsperaFlag(OK);
```

```
                //Obtenemos los datos
                subPuntos[i][j][k]->x = Lee();

                //Confirmamos que hemos leído
                Control(LEIDO);
            }
        }
    }
    Control(0x0);

    //Procesamos la coordenada Y
    //Primero ajustamos el canal de salida
    AjustaEscritura();

    //Uno a uno volcamos los puntos
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            //Fijamos los datos
            Escribe(puntos[i][j]->y);

            //Ajustamos las señales
            Control(SET);

            //Esperamos a que haya grabado
            EsperaFlag(OK);

            //Ajustamos las señales de control
            Control(ESCRITO);
        }
    }

    //Negamos todo
    Control(0x0);

    //Nos preparamos para leer
    AjustaLectura();

    //Uno a uno leemos todos los datos
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            for (int k = 0; k < 4; k++) {
```

```
                //Activamos GET
                Control(GET);

                //Esperamos a que nos diga OK
                EsperaFlag(OK);

                //Obtenemos los datos
                subPuntos[i][j][k]->y = Lee();

                //Confirmamos que hemos leído
                Control(LEIDO);
            }
        }
    }
    Control(0x0);

    //Procesamos la coordenada Z
    //Primero ajustamos el canal de salida
    AjustaEscritura();

    //Uno a uno volcamos los puntos
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            //Fijamos los datos
            Escribe(puntos[i][j]->z);

            //Ajustamos las señales
            Control(SET);

            //Esperamos a que haya grabado
            EsperaFlag(OK);

            //Ajustamos las señales de control
            Control(ESCRITO);
        }
    }

    //Negamos todo
    Control(0x0);

    //Nos preparamos para leer
```

```
    AjustaLectura();

    //Uno a uno leemos todos los datos
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            for (int k = 0; k < 4; k++) {
                //Activamos GET
                Control(GET);

                //Esperamos a que nos diga OK
                EsperaFlag(OK);

                //Obtenemos los datos
                subPuntos[i][j][k]->z = Lee();

                //Confirmamos que hemos leído
                Control(LEIDO);
            }
        }
    }
    Control(0x0);
}

#endif //FPGA
```

B.10. Funciones para el cálculo en punto fijo

B.10.1. pfijo.h

```
#ifndef PFIJO_H
#define PFIJO_H
#include "splines.h"

#ifdef PFIJO_CORTO
typedef unsigned int Long_PFijo;
#else
typedef unsigned long long int Long_PFijo;
#endif

#ifdef PFIJO_TEST
typedef float PFijo;
```

```
#else
#ifdef PFIJO_CORTO
typedef unsigned short int PFijo;
#else
typedef unsigned int PFijo;
#endif //Corto
#endif //Test

float inline a_float(PFijo n) {
    #ifdef PFIJO_TEST
    return n;
    #else
    #ifdef PFIJO_CORTO
    Long_PFijo a = 0x01;
    a <<= 16;
    return (((float)n)/((float)a));
    #else //Corto
    Long_PFijo a = 0x01;
    a <<= 32;
    return (((float)n)/((float)a));
    #endif //Corto
    #endif //Test
}

PFijo inline a_fijo(float n) {
    #ifdef PFIJO_TEST
    return n;
    #else
    #ifdef PFIJO_CORTO
    if (n == 1.f) {
        return 0xFFFF;
    } else {
        Long_PFijo a = 0x01;
        a <<= 16;
        return ((PFijo)(n * ((float)a)));
    }
    #else //Corto
    if (n == 1.f) {
        return 0xFFFFFFFF;
    } else {
        Long_PFijo a = 0x01;
```

```
        a <<= 32;
        return ((PFijo)(n * ((float)a)));
    }
    #endif //Corto
    #endif //Test
}

PFijo inline prod(PFijo a, PFijo b) {
    #ifdef PFIJO_TEST
    return (a*b);
    #else
    Long_PFijo al = a;
    Long_PFijo bl = b;
    #ifdef PFIJO_CORTO
    return (PFijo) ((bl * al) >> 16);
    #else //Corto
    return (PFijo) ((bl * al) >> 32);
    #endif //Corto
    #endif //Test
}

PFijo inline suma_fijo(PFijo a, PFijo b) {
    #ifdef PFIJO_TEST
    return (a+b);
    #else
    Long_PFijo al = a;
    Long_PFijo bl = b;
    Long_PFijo ret = al + bl;
    #ifdef PFIJO_CORTO
    if (ret > 0xFFFF)
        ret = 0xFFFF;
    #else //Corto
    if (ret > 0xFFFFFFFF)
        ret = 0xFFFFFFFF;
    #endif //Corto
    return ret;
    #endif //Test
}

#endif
```

B.10.2. pfijo.cpp

```
#include "pfijo.h"

//Este archivo se crea por necesidades de compilación,
//pero no tiene utilidad ninguna.
```

B.11. Clase *Punto*

B.11.1. punto.h

```
#ifndef PUNTO_H
#define PUNTO_H
#include "splines.h"

#include "pfijo.h"

/**
 * Representación de un nodo de control de una malla.
 * @author Mónica Jiménez Antón
 * @author Carlos Piñeiro Cordero
 * @author Cristina Valbuena Lledó
 */
class Punto {
public:
    /**
     * Coordenada x del punto.
     */
    PFijo x;

    /**
     * Coordenada y del punto.
     */
    PFijo y;

    /**
     * Coordenada z del punto.
     */
    PFijo z;

    /**
```

```
    * Constructor por defecto.
    */
   Punto();

   /**
    * Construye un punto dadas sus coordenadas.
    * @param x Coordenada x del punto.
    * @param y Coordenada x del punto.
    * @param z Coordenada x del punto.
    */
   Punto(float x, float y, float z);

   /**
    * Construye un punto dadas sus coordenadas.
    * @param x Coordenada x del punto.
    * @param y Coordenada x del punto.
    * @param z Coordenada x del punto.
    */
   #ifndef PFIJO_TEST
   Punto(PFijo x, PFijo y, PFijo z);
   #endif

   /**
    * Destructor.
    */
   ~Punto();

   #ifndef SPLINES_HARDWARE
   /**
    * Formatea un punto en una cadena de texto que sea
    * legible.
    * @return string con los datos.
    */
   string toString();
   #endif
};

#endif
```

B.11.2. punto.cpp

```
#include "punto.h"

#include <sstream>
//Constructor
Punto::Punto() {
    // NAda
}

//Constructor
Punto::Punto(float x, float y, float z) {
    this->x = a_fijo(x);
    this->y = a_fijo(y);
    this->z = a_fijo(z);
}

#ifndef SPLINES_HARDWARE
//Conversión en string.
string Punto::toString( ) {
    stringstream strm;
    strm << '[' << a_float(x) << ", " << a_float(y)
    << ", " << a_float(z) << ']';
    return strm.str();
}
#endif

//Destructor
Punto::~ Punto() {
    //Nada.
}

#ifndef PFIJO_TEST
Punto::Punto(PFijo x, PFijo y, PFijo z) {
    this->x = x;
    this->y = y;
    this->z = z;
}
#endif
```

B.12. Clase *Recta*

B.12.1. recta.h

```
#ifndef RECTA_H
#define RECTA_H
#include "splines.h"

#include "caja.h"

/**
 * Representación de una recta en forma paramétrica.
 * @author Mónica Jiménez Antón
 * @author Carlos Piñeiro Cordero
 * @author Cristina Valbuena Lledó
 */
class Recta {
private:
    /**
     * Coeficiente en x del parámetro.
     */
    float a;

    /**
     * Término independiente en x.
     */
    float b;

    /**
     * Coeficiente en y del parámetro.
     */
    float c;

    /**
     * Término independiente en y.
     */
    float d;

    /**
     * Coeficiente en z del parámetro.
     */
```

```
    float e;

    /**
     * Término independiente en z.
     */
    float f;

    /**
     * Haz de rectas (del campo de visión).
     */
    static Recta*** hazRectas;

    /**
     * Anchura del campo de visión.
     */
    static int ancho;

    /**
     * Altura del campo de visión.
     */
    static int alto;

public:
    /**
     * Constructor por defecto.
     */
    Recta();

    /**
     * Construye la recta [at+b, ct+d, et+f].
     * @param a Coeficiente en x.
     * @param b Término independiente en x.
     * @param c Coeficiente en y.
     * @param d Término independiente en y.
     * @param e Coeficiente en z.
     * @param f Término independiente en z.
     */
    Recta(float a, float b, float c, float d, float e,
          float f);

    /**
```

```
     * Destructor.
     */
    ~Recta();

    #ifndef SPLINES_HARDWARE
    /**
     * Imprime una recta en un formato legible.
     * @return string con la recta impresa.
     */
    string toString();
    #endif

    /**
     * Comprueba si esta recta corta a una caja dada.
     * @param c Caja con la que se comprueba.
     * @return True si corta, false si no.
     */
    bool corta(Caja &c);

    /**
     * Crea un haz de rectas con tantas rectas de alto
     * y ancho como se indiquen.
     * @param ancho Anchura del haz.
     * @param alto Altura del haz.
     */
    static void creaHazRectas(int ancho, int alto);

    /**
     * Elimina el haz.
     */
    static void limpia();

    /**
     * Permite acceder al haz.
     * @return Recta*** con el haz.
     */
    static Recta inline *** getHaz() {
        return Recta::hazRectas;
    }
};
```

```
#endif
```

B.12.2. recta.cpp

```
#include "recta.h"

Recta*** Recta::hazRectas = 0;
int Recta::ancho = 0;
int Recta::alto = 0;

//Constructor.
Recta :: Recta() {
    //Nada.
}

//Constructor
Recta::Recta( float a, float b, float c, float d,
              float e, float f ) {
    this->a = a;
    this->b = b;
    this->c = c;
    this->d = d;
    this->e = e;
    this->f = f;
}

#ifndef SPLINES_HARDWARE
//Convierte en un string.
string Recta::toString( ) {
    stringstream strm;
    strm << '[';
    strm << this->a << "·t + " << this->b << ", ";
    strm << this->c << "·t + " << this->d << ", ";
    strm << this->e << "·t + " << this->f << ']';
    return strm.str();
}
#endif

//Comprueba si la recta corta a la caja.
bool Recta :: corta(Caja &c) {
    //Despejamos el parámetro en z.
```

```
    float Z0 = 2-a_float(c.getMax().z);
    float Z1 = 2-a_float(c.getMin().z);

    //Despejamos el parámetro en y.
    float Y0 = (a_float(c.getMin().y) - d) / this->c;
    float Y1 = (a_float(c.getMax().y) - d) / this->c;
    //Ponemos en y0 el menor y en y1 el mayor.
    SWAP(Y0, Y1);

    //Despejamos el parámetro en x.
    float X0 = (a_float(c.getMin().x) - b) / a;
    float X1 = (a_float(c.getMax().x) - b) / a;
    //Ponemos en x0 el menor y en x1 el mayor.
    SWAP(X0, X1);

    //Obtenemos el máximo de los mínimos.
    float max = MAX(X0, Y0);
    max = MAX(max, Z0);

    //Obtenemos el mínimo de los máximos.
    float min = MIN(X1, Y1);
    min = MIN(min, Z1);

    //Si se cruzan, la caja y la recta se cortan.
    return max < min;
}

//Crea el haz de rectas.
void Recta::creaHazRectas(int anchoInt, int altoInt) {
    if (!hazRectas) {
        Recta::ancho = anchoInt;
        Recta::alto = altoInt;
        hazRectas = new Recta**[anchoInt];
        float ancho = (float) anchoInt;
        float alto = (float) altoInt;
        float anchoD = 1.f / (ancho * 2.f);
        float altoD = 1.f / (alto * 2.f);

        #ifdef SPLINES_HARDWARE
        print("Computando rectas...\r\n");
        #endif
```

```
        for (int i = 0; i < anchoInt; i++) {
            hazRectas[i] = new Recta*[altoInt];
            for (int j = 0; j < altoInt; j++) {
                hazRectas[i][j] = new Recta(
                                        ((float)i)/ancho
                                        + anchoD - 0.5f, 0.5f,
                                        ((float)j)/alto
                                        + altoD - 0.5f, 0.5f,
                                        -1.f,2.f);
            }
            #ifdef SPLINES_HARDWARE
            print(".");
            #endif
        }
    }
}

//Limpia el haz de rectas.
void Recta::limpia( ) {
    if (hazRectas) {
        for (int i = 0; i < ancho; i++) {
            for (int j = 0; j < alto; j++) {
                delete hazRectas[i][j];
            }
            delete[] hazRectas[i];
        }
        delete[] hazRectas;
    }
}

Recta::~ Recta() {
    //Nada.
}
```

B.13. Clase *Vec*

B.13.1. vec.h

```
#ifndef VEC_H
#define VEC_H
#include "splines.h"
```

```
#include "pfijo.h"
#include "punto.h"

/**
 * Representación de un vector.
 * @author Mónica Jiménez Antón
 * @author Carlos Piñeiro Cordero
 * @author Cristina Valbuena Lledó
 */
class Vec {
public:
    /**
     * Coordenada i del vector.
     */
    float i;

    /**
     * Coordenada j del vector.
     */
    float j;

    /**
     * Coordenada k del vector.
     */
    float k;

    /**
     * Constructor por defecto.
     */
    Vec();

    /**
     * Constructor paramétrico
     * @param i Coordenada i del vector.
     * @param j Coordenada j del vector.
     * @param k Coordenada k del vector.
     */
    Vec(float i, float j,float k);

    /**
```

```
     * Calcula el vector normal a un plano dados 3
     * puntos del mismo.
     * @param p1 Punto del plano.
     * @param p2 Punto del plano.
     * @param p3 Punto del plano.
     */
    Vec(Punto *p1, Punto *p2, Punto *p3);

    /**
     * Destructor.
     */
    ~Vec();

    #ifndef SPLINES_HARDWARE
    /**
     * Imprime un vector en un formato legible.
     * @return string con el vector impreso.
     */
    string toString();
    #endif //SPLINES_HARDWARE

    /**
     * Realiza la diferencia entre las coordenadas de
     * this y de v.
     * @param v Vector que se resta.
     */
    void resta(const Vec& v) {
        i -= v.i;
        j -= v.j;
        k -= v.k;
    }

    /**
     * Producto escalar.
     * @param v Vector con el que se realiza el producto.
     * @return float con el producto.
     */
    float inline escalar(const Vec& v) {
        return i * v.i + j * v.j + k * v.k;
    }
```

```
/**
 * Multiplica el módulo del vector por un escalar.
 * @param c Escalar.
 */
void inline multModulo(float c) {
    i *= c;
    k *= c;
    k *= c;
}

/**
 * Calcula el módulo de un vector
 * @return float con el módulo.
 */
float inline modulo() {
    return sqrt(i*i + j*j + k*k);
}

/**
 * Copia los datos de un vector dado.
 * @param v Vector original.
 */
void inline set(Vec &v) {
    this->i = v.i;
    this->j = v.j;
    this->k = v.k;
}

/**
 * Normaliza el vector.
 */
void inline norm() {
    float mod = modulo();
    i /= mod;
    j /= mod;
    k /= mod;
}

/**
 * Invierte el sentido del vector.
 */
```

```
    void inline invertir() {
        i = -(i);
        j = -(j);
        k = -(k);
    }
};

#endif
```

B.13.2. vec.cpp

```
#include "vec.h"

//Constructor
Vec::Vec() {
    //Nada
}

//Constructor.
Vec::Vec(float i, float j,float k) {
    this->i = i;
    this->j = j;
    this->k = k;
}

//Destructor
Vec::~Vec() {
    //Nada
}

#ifndef SPLINES_HARDWARE
//Paso a string.
string Vec:: toString() {
    stringstream strm;
    strm << '[' << this->i << ", " << this->j <<","
    << this->k <<']';
    return strm.str();
}
#endif

//Constructor "normal"
```

```
Vec::Vec(Punto * p1, Punto * p2, Punto * p3) {
    //Calculamos los vectores que unen p2-p1 y p3-p1
    float x1 = a_float(p1->x);
    float x2 = a_float(p2->x);
    float x3 = a_float(p3->x);
    float y1 = a_float(p1->y);
    float y2 = a_float(p2->y);
    float y3 = a_float(p3->y);
    float z1 = a_float(p1->z);
    float z2 = a_float(p2->z);
    float z3 = a_float(p3->z);
    float i1 = x3 - x1;
    float i2 = x2 - x1;
    float j1 = y3 - y1;
    float j2 = y2 - y1;
    float k1 = z3 - z1;
    float k2 = z2 - z1;

    //Ahora calculamos su producto vectorial.
    this->i = j1 * k2 - k1 * j2;
    this->j = k1 * i2 - i1 * k2;
    this->k = i1 * j2 - j1 * i2;

    //Finalmente normalizamos
    this->norm();
}
```

Apéndice C

Listado de código JAVA

El presente apéndice contiene el listado de código en Java que utilizamos para la generación de VHDL.

C.1. Generación del diseño inicial

C.1.1. Clase *GeneraVHDL*

```
package app;

import java.io.FileOutputStream;
import java.io.IOException;

public class GeneraVHDL {

    public static double[][] b;

    public static String entradas;

    public static String salidas;

    public static String señales;

    public static String codigo;

    /**
     *
```

```
     * @param args
     * @throws IOException
     */
    public static void main(String[] args)
            throws IOException {

        // Archivo y ruta donde grabamos el código vhdl
        FileOutputStream archivo = new FileOutputStream(
        "C:\\Documents and Settings\\Escritorio
          \\subdivision.vhd");
        // Inicialización del array con
        // los valores de Bernstein
        GeneraVHDL.iniciaBernstein();

        // Comienzo de los strings
        entradas = "PORT (";
        salidas = "";
        señales = "signal ";
        codigo = "";

        // Entradas de la entidad:
        for (int i = 0; i < 4; i++)
            for (int j = 0; j < 4; j++)
                entradas += "punto"
                + i
                + ""
                + j
                + ": IN std_logic_vector (31 downto 0);\n";

        // Creación de cada malla y sus señales:
        for (int i = 0; i < 4; i++) {
            creaMalla(i);
            señales += ((i < 3) ? " \n" : "");
        }

        // Final del string de señales
        señales += ": std_logic_vector (31 downto 0);\n";

        // Impresión con print
        System.out.println("library IEEE;\n"
                + "use IEEE.STD_LOGIC_1164.ALL;\n"
```

```
            + "use IEEE.STD_LOGIC_ARITH.ALL;\n"
            + "use IEEE.STD_LOGIC_UNSIGNED.ALL;\n");
        System.out.println("entity subdivision is\n");
        System.out.println("--ENTRADAS: \n" + entradas);
        System.out.println("--SALIDAS: \n" + salidas);
        System.out.println("end subdivision;
          \n \n architecture Behavioral of "
                  + "subdivision is\n");
        System.out.println("--SEÑALES: \n" + señales);
        System.out.println("begin\n");
        System.out.println("--CODIGO: \n" + codigo);
        System.out.println("\n\nend Behavioral;");

        // Impresión en archivo
        String imprimir = "library IEEE;\n"
            + "use IEEE.STD_LOGIC_1164.ALL;\n"
            + "use IEEE.STD_LOGIC_ARITH.ALL;\n"
            + "use IEEE.STD_LOGIC_UNSIGNED.ALL;\n"
            + "entity subdivision is\n"
            + "--ENTRADAS: \n"
            + entradas
            + "--SALIDAS: \n"
            + salidas
            + "end subdivision;
                \n \narchitecture Behavioral of "
            + "subdivision is\n" + "--SEÑALES: \n"
            + señales + "begin\n" + "--CODIGO: \n"
            + codigo + "\n\nend Behavioral;";
        archivo.write(imprimir.getBytes());
        archivo.close();

    }

    /**
     * Método para inicializar el array b
     * con los valores de Bernstein
     *
     */
    private static void iniciaBernstein() {
        GeneraVHDL.b = new double[4][4];
```

```
        b[0][0] = 1;
        b[0][1] = 0;
        b[0][2] = 0;
        b[0][3] = 0;
        b[1][0] = 0.5;
        b[1][1] = 0.5;
        b[1][2] = 0;
        b[1][3] = 0;
        b[2][0] = 0.25;
        b[2][1] = 0.5;
        b[2][2] = 0.25;
        b[2][3] = 0;
        b[3][0] = 0.125;
        b[3][1] = 0.375;
        b[3][2] = 0.375;
        b[3][3] = 0.125;
    }

    /**
     * Generación de cada nodo intermedio de una malla
     *
     * @param malla
     */
    public static void creaMalla(int malla) {

        codigo += "--------------------";
        codigo += "--   Malla " + malla;
        codigo += "--------------------\n";

        for (int i = 0; i < 4; i++)
            for (int j = 0; j < 4; j++) {
                switch (malla) {
                case 0:
                    GeneraVHDL.nodoIntermedio(malla, 0, 0,
                            j, i);
                    break;
                case 1:
                    GeneraVHDL.nodoIntermedio(malla, 0, i,
                            j, 3 - i);
                    break;
                case 2:
```

```
                GeneraVHDL.nodoIntermedio(malla, j, 0,
                        3 - j, i);
                break;
            case 3:
                GeneraVHDL.nodoIntermedio(malla, j, i,
                        3 - j, 3 - i);
                break;
            default:
                break;
            }
        }
    }

    /**
     * Nombra un nodo intermedio según los parámetros
     *
     * @param malla
     * @param i
     * @param j
     * @param r
     * @param s
     */
    public static void nodoIntermedio(int malla, int i,
            int j, int r, int s) {

        String nodo = "";

        switch (malla) {
        case 0:
            nodo += "N" + s + "" + r;
            break;
        case 1:
            nodo += "N" + j + "" + r;
            break;
        case 2:
            nodo += "N" + s + "" + i;
            break;
        case 3:
            nodo += "N" + j + "" + i;
            break;
        default:
```

```
            break;
        }
        GeneraVHDL.calculaNodo(malla, nodo, r, s, i, j);
    }

    /**
     * Calcula los sumandos de un nodo según
     * los parámetros y los almacena en el
     * string adecuado.
     *
     * @param malla
     * @param nodo
     * @param r
     * @param s
     * @param i
     * @param j
     */
    private static void calculaNodo(int malla,
            String nodo, int r, int s, int i, int j) {

    // Nodo de una submalla del que calculamos el valor:
        String cadena = "malla" + malla + "" + nodo
                + " <= ";
        String cadena2 = "";

        // Añadimos al string de salidas el nuevo nodo
        // de una de las submallas:
        salidas += "malla"
                + malla
                + ""
                + nodo
                + ": OUT std_logic_vector (31 downto 0)"
                + (((malla == 3) && nodo
                        .equalsIgnoreCase("N33")) ? ");"
                        : ";") + "\n ";

        for (int k = 0; k <= r; k++) {
            for (int l = 0; l <= s; l++) {
                if (b[s][l] != 0 && b[r][k] != 0) {
                    // si el sumando no se anula:
                    // añadimos el sumando al string:
```

```
                cadena += "m" + malla + nodo
                      + "sumando" + k + "" + l;
                // añadimos el sumando al string
                // de señales intermedias:
                señales += "m"
                      + malla
                      + nodo
                      + "sumando"
                      + k
                      + ""
                      + l
                      + ((k == 0 && l == 0
                          && malla == 3 && nodo
                          .equalsIgnoreCase("N33")) ? ""
                          : ",");

                // calculamos el producto
                // de los dos bernstein:
                double B = b[r][k] * b[s][l];
                if (B != 1) {
                // Si hay que multiplicar
                // ( si el resultado no es 1):
                   cadena2 += generaCadena(B, i, k, j,
                         l, malla, nodo);
                } else {
                   // Si el sumando es sólo el punto:
                   cadena2 += "m" + malla + nodo
                         + "sumando" + k + "" + l
                         + " <= " + " punto" + (i + k)
                         + "" + (j + l) + ";\n";
                }
                // Completamos la cadena según sea
                // el último sumando o uno intermedio
                cadena += (((k == r && l == s) ? ";\n"
                      : " + "));
          }
       }
    }

    // Añade todos los resultados intermedios:
    codigo += cadena2;
```

```
        // Añade la suma:
        codigo += cadena + '\n';
    }

    /**
     * Genera un sumando de un nodo de la malla
     * transformando todas las multiplicaciones
     * en desplazamientos o desplazamientos y sumas
     *
     * @param B
     * @param i
     * @param k
     * @param j
     * @param l
     * @param malla
     * @param nodo
     * @return string con la cadena de operaciones
     */
    private static String generaCadena(double B, int i,
            int k, int j, int l, int malla, String nodo) {

        String cadena = "";

        // Comprobamos si se puede sustituir por un
        // desplazamiento o varios:
        String bin = GeneraVHDL.doubleToString(B);
        int desplazar = 0;
        int numDesplaz1 = -1;
        int numDesplaz2 = -1;
        int aux = 1;
        while (aux < bin.length() && (desplazar <= 2)) {
            if (bin.charAt(aux) == '1') {
                desplazar++;
                // si es el primer desplazamiento
                if (numDesplaz1 == -1)
                    // guardamos la posición del primer 1
                    numDesplaz1 = aux;
                else
                    // guardamos la posición del segundo 1
                    numDesplaz2 = aux;
            }
```

```
            aux++;
        }

        // Desplazamos una vez:
        if (desplazar < 2) {
            // Creamos una cadena de ceros, tantos como
            // la posición en la que se encontraba el uno
            // más uno
            String ceros = "";
            for (int v = 0; v < numDesplaz1; v++)
                ceros += "0";
            // Añadimos a la cadena el sumando y su valor:
            // su valor es el desplazamiento del punto.
            cadena += "m" + malla + nodo + "sumando" + k
                    + "" + l + " <= " + "\"" + ceros + "\""
                    + " & punto" + (i + k) + "" + (j + l)
                    + "(31 downto "
                    + (numDesplaz1 /* + 1 */) + ");\n";

        } else {
            // Desplazamos dos veces:
            String ceros = "";
            // Creamos una cadena de ceros, tantos como
            // la posición en la que se encontraba
            // el primer uno más uno
            for (int v = 0; v < numDesplaz1; v++)
                ceros += "0";
            // Añadimos a la cadena el valor parcial
            // del sumando:
            cadena += "m" + malla + nodo + "sumando" + k
                    + "" + l + " <=( " + "\"" + ceros
                    + "\"" + " & punto" + (i + k) + ""
                    + (j + l) + "(31 downto "
                    + (numDesplaz1 /* + 1 */) + "))";
            ceros = "";
            // Volvemos a crear una cadena de ceros,
            // tantos como la posición en la que
            // se econtraba el segundo más uno
            for (int v = 0; v < numDesplaz2; v++)
                ceros += "0";
            // Añadimos a la cadena la suma que le faltaba
```

```
            // al segundo con el valor del punto
            // desplazado según el segundo 1.
            cadena += "+ (" + "\"" + ceros + "\""
                  + " & punto" + (i + k) + "" + (j + l)
                  + "(31 downto "
                  + (numDesplaz2 /* + 1 */) + "));\n";

         }

         return cadena;
      }

      /**
       * Paso de double a binario
       *
       * @param b: número en double
       * @return: número en binario
       */
      private static String doubleToString(double b) {
         String retorno = "";
         long a = 0x1;
         a = a << 32;
         long result = (long) (((float) a) * b);
         retorno = Long.toBinaryString(result);
         while (retorno.length() < 32)
            retorno = "0" + retorno;
         retorno = "\"" + retorno + "\"";
         return retorno;
      }
}
```

C.2. Generación de código de un diseño reducido

C.2.1. Clase *GeneraVHDL*

```
package app;

import java.util.ArrayList;
import java.util.HashSet;
```

```
public class GeneraVHDL {

   private static double[][] b;

   private static ArrayList<String> senales =
                        new ArrayList<String>();

   private static HashSet<Suma> sumandos =
                         new HashSet<Suma>();

   private static HashSet<Desplazamiento>
   desplazamientos = new HashSet<Desplazamiento>();

   private static String codigoDesplazamientos = "";

   private static String codigoSumandos = "";

   private static String codigoSalida = "";

   public static void main(String[] args) {
      // Iniciamos bernstein
      GeneraVHDL.iniciaBernstein();

      // Creación de cada malla y sus señales:
      for (int i = 0; i < 4; i++) {
         creaMalla(i);
      }

      System.out.println(codigoDesplazamientos);
      System.out.println(codigoSumandos);
      System.out.println(codigoSalida);

      // Señales
      System.out.print("signal " + senales.get(0));
      for (int i = 1; i < senales.size(); i++) {
         System.out.print(", ");
         if (i %5 == 0)
            System.out.println();
         System.out.print(senales.get(i));
      }
```

```
        System.out.println(" :
                        std_logic_vector(31 downto 0);");
    }

    /**
     * Generación de cada nodo intermedio de una malla
     *
     * @param malla
     */
    public static void creaMalla(int malla) {

        for (int i = 0; i < 4; i++) {
            for (int j = 0; j < 4; j++) {
                switch (malla) {
                case 0:
                    GeneraVHDL.nodoIntermedio(malla, 0, 0,
                            j, i);
                    break;
                case 1:
                    GeneraVHDL.nodoIntermedio(malla, 0, i,
                            j, 3 - i);
                    break;
                case 2:
                    GeneraVHDL.nodoIntermedio(malla, j, 0,
                            3 - j, i);
                    break;
                case 3:
                    GeneraVHDL.nodoIntermedio(malla, j, i,
                            3 - j, 3 - i);
                    break;
                default:
                    break;
                }
            }
        }
    }

    /**
     * Nombra un nodo intermedio según los parámetros
     *
     * @param malla
```

```
     * @param i
     * @param j
     * @param r
     * @param s
     */
    public static void nodoIntermedio(int malla, int i,
            int j, int r, int s) {

        String nodo = "";

        switch (malla) {
        case 0:
            nodo += "N" + s + "" + r;
            break;
        case 1:
            nodo += "N" + j + "" + r;
            break;
        case 2:
            nodo += "N" + s + "" + i;
            break;
        case 3:
            nodo += "N" + j + "" + i;
            break;
        default:
            break;
        }
        GeneraVHDL.calculaNodo(malla, nodo, r, s, i, j);
    }

    /**
     * Calcula los sumandos de un nodo según
     * los parámetros y los almacena en el
     * string adecuado.
     *
     * @param malla
     * @param nodo
     * @param r
     * @param s
     * @param i
     * @param j
     */
```

```
private static void calculaNodo(int malla,
      String nodo, int r, int s, int i, int j) {

   Suma suma = new Suma();
   for (int k = 0; k <= r; k++) {
      for (int l = 0; l <= s; l++) {
         // Obtenemos los desplazamientos
         Desplazamiento[] sumandos =
         generaDesplazamientos(b[r][k] * b[s][l],
                               i, k, j, l);

         // Insertamos los desplazamientos
         GeneraVHDL
               .insertaDesplazamientos(sumandos);

         // Añadimos los desplazamientos a la suma
         for (int m = 0; m < sumandos.length; m++)
            suma.sumandos.add(sumandos[m]);
      }
   }

   String nombreSenal = "";
   // Comprobamos si existen los sumandos
   if (GeneraVHDL.sumandos.contains(suma)) {
      // Entonces obtenemos el nombre de la señal.
      nombreSenal = GeneraVHDL.obtieneSuma(suma);
   } else {
      // Hay que crear la señal.
      nombreSenal = GeneraVHDL.creaSuma(suma);
   }
   // Generamos el código de salida
   codigoSalida += "malla" + malla + nodo + " <= "
         + nombreSenal + ";\n";
}

private static String obtieneSuma(Suma s) {
   String ret = null;
   Suma[] sums = new Suma[GeneraVHDL.sumandos
         .size()];
   GeneraVHDL.sumandos.toArray(sums);
   for (int i = 0; i < sums.length &&
```

```
                                        ret == null; i++)
            if (sums[i].equals(s))
                ret = sums[i].nombre;
        return ret;
    }

    private static String creaSuma(Suma suma) {
        // Lo primero es obtener el nombre de la
        // que será la señal. Para ello,
        // obtenemos los desplazamientos que la componen
        Desplazamiento[] desps =
                new Desplazamiento[suma.sumandos.size()];
        suma.sumandos.toArray(desps);

        // Ahora creamos el nombre de la señal
        String nombre = "suma";
        for (int i = 0; i < desps.length; i++) {
            nombre += "_" + desps[i].ceros + ""
                    + desps[i].x + "" + desps[i].y;
        }

        // Creamos el código
        String codigo = nombre
                + " <= "
                + GeneraVHDL
                        .obtieneDesplazamiento(desps[0]);
        for (int i = 1; i < desps.length; i++)
            codigo += " + "
                    + GeneraVHDL
                            .obtieneDesplazamiento(desps[i]);
        codigo += ";\n";

        // Guardamos el código
        GeneraVHDL.codigoSumandos += codigo;

        // Finalmente almacenamos el nombre y el sumando
        suma.nombre = nombre;
        GeneraVHDL.sumandos.add(suma);

        // Almacenamos el nombre de la señal
        GeneraVHDL.senales.add(nombre);
```

```
        // Devolvemos el nombre;
        return nombre;
    }

    private static String obtieneDesplazamiento(
            Desplazamiento d) {
        String ret = null;
        Desplazamiento[] desps =
new Desplazamiento[GeneraVHDL.desplazamientos.size()];
        GeneraVHDL.desplazamientos.toArray(desps);
        for (int i = 0; i < desps.length &&
                ret == null; i++)
            if (desps[i].equals(d))
                ret = desps[i].nombre;
        return ret;
    }

    private static void insertaDesplazamientos(
            Desplazamiento[] sumandos) {
        // Para cada desplazamiento
        for (int i = 0; i < sumandos.length; i++) {
            // Comprobamos si existe
            if (!GeneraVHDL.desplazamientos
                    .contains(sumandos[i])) {
                // Entonces lo añade
                String cadena = "puntoDes"
                        + sumandos[i].ceros + ""
                        + sumandos[i].x + "" + sumandos[i].y;
                sumandos[i].nombre = cadena;
                GeneraVHDL.desplazamientos
                        .add(sumandos[i]);

                // Guardamos el nombre de la señal
                GeneraVHDL.senales.add(cadena);

                // Y generamos el código
                cadena += " <= ";
                if (sumandos[i].ceros != 0) {
                  cadena += "\"";
                  for(int j = 0; j< sumandos[i].ceros; j++)
```

```
                    cadena += "0";
                cadena += "\" &  punto" + sumandos[i].x
                        + "" + sumandos[i].y
                        + "(31 downto ";
                cadena += sumandos[i].ceros + ");\n";
            } else {
                cadena += "punto" + sumandos[i].x + ""
                        + sumandos[i].y + ";\n";
            }

            // Guardamos el código del desplazamiento
            GeneraVHDL.codigoDesplazamientos += cadena;
        }
    }
}

private static Desplazamiento[] generaDesplazamientos(
        double B, int i, int k, int j, int l) {
    Desplazamiento[] ret = null;
    int x = i + k;
    int y = j + l;
    if (B == 0) {
        // No se suma
        ret = new Desplazamiento[0];
    } else if (B == 1) {
        // Se suma el punto tal cual
        ret = new Desplazamiento[1];
        ret[0] = new Desplazamiento();
        ret[0].x = x;
        ret[0].y = y;
        ret[0].ceros = 0;
    } else {
        // Obtenemos la cadena de 1s y 0s
        String des = GeneraVHDL.doubleToString(B);

        // Obtenemos la posición de los 2 posibles 1s.
        int desplazar = 0;
        int numDesplaz1 = -1;
        int numDesplaz2 = -1;
        int aux = 0;
        while(aux < des.length() && (desplazar <= 2)){
```

```
            if (des.charAt(aux) == '1') {
                desplazar++;
                // si es el primer desplazamiento
                if (numDesplaz1 == -1) {
                    // guardamos la posición del primer 1
                    numDesplaz1 = aux;
                } else {
                    //guardamos la posición del segundo 1
                    numDesplaz2 = aux;
                }
            }
            aux++;
        }

        // Miramos cuantos 1s hay
        if (desplazar == 1) {
            ret = new Desplazamiento[1];
            ret[0] = new Desplazamiento();
            ret[0].x = x;
            ret[0].y = y;
            ret[0].ceros = numDesplaz1 + 1;
        } else {
            ret = new Desplazamiento[2];
            ret[0] = new Desplazamiento();
            ret[1] = new Desplazamiento();
            ret[0].x = x;
            ret[0].y = y;
            ret[0].ceros = numDesplaz1 + 1;
            ret[1].x = x;
            ret[1].y = y;
            ret[1].ceros = numDesplaz2 + 1;
        }
    }
    return ret;
}

/**
 * Método para inicializar el array b
 * con los valores de Bernstein
 *
 */
```

```
    private static void iniciaBernstein() {
        GeneraVHDL.b = new double[4][4];
        b[0][0] = 1;
        b[0][1] = 0;
        b[0][2] = 0;
        b[0][3] = 0;
        b[1][0] = 0.5;
        b[1][1] = 0.5;
        b[1][2] = 0;
        b[1][3] = 0;
        b[2][0] = 0.25;
        b[2][1] = 0.5;
        b[2][2] = 0.25;
        b[2][3] = 0;
        b[3][0] = 0.125;
        b[3][1] = 0.375;
        b[3][2] = 0.375;
        b[3][3] = 0.125;
    }

    /**
     * Paso de double a binario
     *
     * @param b:
     *            número en double
     * @return: número en binario
     */
    private static String doubleToString(double b) {
        String retorno = "";
        long a = 0x1;
        a = a << 32;
        long result = (long) (((float) a) * b);
        retorno = Long.toBinaryString(result);
        while (retorno.length() < 32)
            retorno = "0" + retorno;
        retorno = "\"" + retorno + "\"";
        return retorno;
    }

}
```

C.2.2. Clase *Desplazamiento*

```
package app;

/**
 * Desplazamiento. Consiste de un punto de entrada y
 * un número de ceros de desplazamientos.
 */
public class Desplazamiento {
   public int x;

   public int y;

   public int ceros;

   // Nombre de la señal que tiene este desplazamiento
   public String nombre;

   /**
    *  Sobreescribimos la función hash
    *  para poder incluir este objeto
    *  dentro de un hashSet
    */
   public int hashCode() {
      return x + y * 10 + ceros * 100;
   }

   public boolean equals(Object ob) {
      if (ob.getClass() != this.getClass())
         return false;
      Desplazamiento o = (Desplazamiento) ob;
      return o.x == this.x && o.y == this.y
            && o.ceros == this.ceros;
   }
}
```

C.2.3. Clase *Suma*

```
package app;
```

```
import java.util.HashSet;

/**
 * Elemento "suma", que suma un conjunto
 * de desplazamientos.
 */
public class Suma {
   public HashSet<Desplazamiento> sumandos =
                    new HashSet<Desplazamiento>();

   public String nombre;

   public int hashCode() {
      int suma = 0;
      Object[] sums = sumandos.toArray();
      for (int i = 0; i < sums.length; i++)
         suma += sums[i].hashCode();
      return suma;
   }

   public boolean equals(Object ob) {
      if (ob.getClass() != this.getClass())
         return false;
      Suma o = (Suma) ob;
      return o.sumandos.equals(sumandos);
   }
}
```

C.3. Generación de código del diseño en árbol

C.3.1. Clase *GeneraVHDL*

```
package app;

import java.util.ArrayList;
import java.util.HashSet;
import java.util.Iterator;

public class GeneraVHDL {
```

```
    private static double[][] b;

    private static ArrayList<String> senales =
                          new ArrayList<String>();

    private static HashSet<Suma> sumandos =
                          new HashSet<Suma>();

    private static HashSet<Desplazamiento>
       desplazamientos = new HashSet<Desplazamiento>();

    private static ArrayList<String>
       codigoDesplazamientos = new ArrayList<String>();

    private static ArrayList<String> codigoSumandos =
                           new ArrayList<String>();

    private static ArrayList<String> codigoSalida =
                           new ArrayList<String>();

    public static void main(String[] args) {
       // Iniciamos bernstein
       GeneraVHDL.iniciaBernstein();

       // Creación de cada malla y sus señales:
       for (int i = 0; i < 4; i++) {
          creaMalla(i);
       }

       Iterator<String> i =
                         GeneraVHDL.codigoDesplazamientos
             .iterator();
       while (i.hasNext())
          System.out.println(i.next());
       System.out.println();

       i = GeneraVHDL.codigoSumandos.iterator();
       while (i.hasNext())
          System.out.println(i.next());
       System.out.println();
```

```
        i = GeneraVHDL.codigoSalida.iterator();
        while (i.hasNext())
            System.out.println(i.next());

        // Señales
        System.out.print("signal ");
        i = GeneraVHDL.senales.iterator();
        System.out.print(i.next());
        while (i.hasNext())
            System.out.print(", " + i.next());
        System.out.println
                (" : std_logic_vector(31 downto 0);");
    }

    /**
     * Generación de cada nodo intermedio de una malla
     *
     * @param malla
     */
    public static void creaMalla(int malla) {

        for (int i = 0; i < 4; i++) {
            for (int j = 0; j < 4; j++) {
                switch (malla) {
                case 0:
                    GeneraVHDL.nodoIntermedio(malla, 0, 0,
                            j, i);
                    break;
                case 1:
                    GeneraVHDL.nodoIntermedio(malla, 0, i,
                            j, 3 - i);
                    break;
                case 2:
                    GeneraVHDL.nodoIntermedio(malla, j, 0,
                            3 - j, i);
                    break;
                case 3:
                    GeneraVHDL.nodoIntermedio(malla, j, i,
                            3 - j, 3 - i);
                    break;
                default:
```

```
                break;
            }
        }
    }
}

/**
 * Nombra un nodo intermedio según los parámetros
 *
 * @param malla
 * @param i
 * @param j
 * @param r
 * @param s
 */
public static void nodoIntermedio(int malla, int i,
        int j, int r, int s) {

    String nodo = "";

    switch (malla) {
    case 0:
        nodo += "N" + s + "" + r;
        break;
    case 1:
        nodo += "N" + j + "" + r;
        break;
    case 2:
        nodo += "N" + s + "" + i;
        break;
    case 3:
        nodo += "N" + j + "" + i;
        break;
    default:
        break;
    }
    GeneraVHDL.calculaNodo(malla, nodo, r, s, i, j);
}

/**
 * Calcula los sumandos de un nodo según
```

```
     * los parámetros y los almacena en el
     * string adecuado.
     *
     * @param malla
     * @param nodo
     * @param r
     * @param s
     * @param i
     * @param j
     */
    private static void calculaNodo(int malla,
            String nodo, int r, int s, int i, int j) {

        Suma suma = new Suma();
        for (int k = 0; k <= r; k++) {
            for (int l = 0; l <= s; l++) {
                // Obtenemos los desplazamientos
                Desplazamiento[] sumandos =
                        generaDesplazamientos(
                         b[r][k] * b[s][l], i, k, j, l);

                // Insertamos los desplazamientos
                GeneraVHDL
                        .insertaDesplazamientos(sumandos);

                // Añadimos los desplazamientos a la suma
                for (int m = 0; m < sumandos.length; m++)
                    suma.sumandos.add(sumandos[m]);
            }
        }

        String nombreSenal = "";
        // Comprobamos si existen los sumandos
        if (GeneraVHDL.sumandos.contains(suma)) {
            // Entonces obtenemos el nombre de la señal.
            nombreSenal = GeneraVHDL.obtieneSuma(suma);
        } else {
            // Hay que crear la señal.
            nombreSenal = GeneraVHDL.creaArbolSumas(suma);
        }
        // Generamos el código de salida
```

```
        codigoSalida.add(new String("malla" + malla
                + nodo + " <= " + nombreSenal + ";"));
    }

    private static String obtieneSuma(Suma s) {
        String ret = null;
        Iterator<Suma> i = GeneraVHDL.sumandos
                .iterator();
        while (i.hasNext() && ret == null) {
            Suma sum = i.next();
            if (sum.equals(s)) {
                ret = sum.nombre;
            }
        }
        return ret;
    }

    /**
     * Dada una suma, genera un árbol de sumandos
     * y lo va guardando.
     * @param suma
     * @return nombre de la señal del sumando que recibe.
     */
    private static String creaArbolSumas(Suma suma) {
        String arbol = null;
        if (suma.sumandos.size() > 1) {
            // Dividimos los sumandos en 2.
            // Para ello, primero volcamos los
            // sumandos en un array.
            Desplazamiento[] sumandos =
                    new Desplazamiento[suma.sumandos.size()];
            suma.sumandos.toArray(sumandos);

            // Partimos el array en 2
            int mitad = sumandos.length / 2;
            Desplazamiento[] sum1 = new Desplazamiento[mitad];
            Desplazamiento[] sum2 =
            new Desplazamiento[sumandos.length - mitad];
            // Para
            // el
            // caso
```

```
// impar
for (int i = 0; i < mitad; i++)
   sum1[i] = sumandos[i];
for (int i = mitad; i < sumandos.length; i++)
   sum2[i - mitad] = sumandos[i];

// Metemos los arrays en dos sumas
Suma suma1 = new Suma();
Suma suma2 = new Suma();
for (int i = 0; i < sum1.length; i++)
   suma1.sumandos.add(sum1[i]);
for (int i = 0; i < sum2.length; i++)
   suma2.sumandos.add(sum2[i]);

// Obtenemos los nombres de estos 2 árboles.
String arbol1 = null;
if (GeneraVHDL.sumandos.contains(suma1)) {
   arbol1 = GeneraVHDL.obtieneSuma(suma1);
} else {
   arbol1 = GeneraVHDL.creaArbolSumas(suma1);
}
String arbol2 = null;
if (GeneraVHDL.sumandos.contains(suma2)) {
   arbol2 = GeneraVHDL.obtieneSuma(suma2);
} else {
   arbol2 = GeneraVHDL.creaArbolSumas(suma2);
}

// Creamos ahora una señal que sume
// los 2 subárboles. Primero le
// damos nombre a esta suma
suma.creaNombre();

// Guardamos la suma
GeneraVHDL.sumandos.add(suma);

// Finalmente generamos el código
GeneraVHDL.codigoSumandos.add(suma.nombre
      + " <= " + arbol1 + " " + arbol2
      + ";");
```

```
            // El nombre de este árbol es el nombre de
            // la señal en cuestión
            arbol = suma.nombre;

            // Por último, almacenamos esta señal
            GeneraVHDL.senales.add(arbol);
        } else {
            // Caso base: el árbol sólo tiene un sumando.
            // Buscamos el nombre del desplazamiento.
            arbol = GeneraVHDL
                    .obtieneDesplazamiento(suma.sumandos
                            .iterator().next());
        }

        return arbol;
    }

    private static String obtieneDesplazamiento(
            Desplazamiento d) {
        String ret = null;
        Desplazamiento[] desps =
          new Desplazamiento[GeneraVHDL.desplazamientos
              .size()];
        GeneraVHDL.desplazamientos.toArray(desps);
        for (int i = 0; i < desps.length && ret == null;
             i++)
            if (desps[i].equals(d))
                ret = desps[i].nombre;
        return ret;
    }

    private static void insertaDesplazamientos(
            Desplazamiento[] sumandos) {
        // Para cada desplazamiento
        for (int i = 0; i < sumandos.length; i++) {
            // Comprobamos si existe
            if (!GeneraVHDL.desplazamientos
                    .contains(sumandos[i])) {
                // Entonces lo añade
                String cadena = "puntoDes"
                        + sumandos[i].ceros + ""
```

```
                    + sumandos[i].x + "" + sumandos[i].y;
            sumandos[i].nombre = cadena;
            GeneraVHDL.desplazamientos
                    .add(sumandos[i]);

            // Guardamos el nombre de la señal
            GeneraVHDL.senales.add(cadena);

            // Y generamos el código
            cadena += " <= ";
            if (sumandos[i].ceros != 0) {
               cadena += "\"";
               for (int j = 0; j < sumandos[i].ceros;
                        j++)
                  cadena += "0";
               cadena += "\" &  regE" + sumandos[i].x
                     + "" + sumandos[i].y
                     + "(31 downto ";
               cadena += sumandos[i].ceros + ");";
            } else {
               cadena += "regE" + sumandos[i].x + ""
                     + sumandos[i].y + ";";
            }

            // Guardamos el código del desplazamiento
            GeneraVHDL.codigoDesplazamientos
                    .add(cadena);
         }
      }
   }

   private static Desplazamiento[]
         generaDesplazamientos(double B, int i, int k,
                               int j, int l) {
      Desplazamiento[] ret = null;
      int x = i + k;
      int y = j + l;
      if (B == 0) {
         // No se suma
         ret = new Desplazamiento[0];
      } else if (B == 1) {
```

```
            // Se suma el punto tal cual
            ret = new Desplazamiento[1];
            ret[0] = new Desplazamiento();
            ret[0].x = x;
            ret[0].y = y;
            ret[0].ceros = 0;
        } else {
            // Obtenemos la cadena de 1s y 0s
            String des = GeneraVHDL.doubleToString(B);

            // Obtenemos la posición de los 2 posibles 1s.
            int desplazar = 0;
            int numDesplaz1 = -1;
            int numDesplaz2 = -1;
            int aux = 0;
            while(aux < des.length() && (desplazar <= 2)){
                if (des.charAt(aux) == '1') {
                    desplazar++;
                    // si es el primer desplazamiento
                    if (numDesplaz1 == -1) {
                        // guardamos la posición del primer 1
                        numDesplaz1 = aux;
                    } else {
                        //guardamos la posición del segundo 1
                        numDesplaz2 = aux;
                    }
                }
                aux++;
            }

            // Miramos cuantos 1s hay
            if (desplazar == 1) {
                ret = new Desplazamiento[1];
                ret[0] = new Desplazamiento();
                ret[0].x = x;
                ret[0].y = y;
                ret[0].ceros = numDesplaz1 + 1;
            } else {
                ret = new Desplazamiento[2];
                ret[0] = new Desplazamiento();
                ret[1] = new Desplazamiento();
```

```
                ret[0].x = x;
                ret[0].y = y;
                ret[0].ceros = numDesplaz1 + 1;
                ret[1].x = x;
                ret[1].y = y;
                ret[1].ceros = numDesplaz2 + 1;
            }
        }
        return ret;
    }

    /**
     * Método para inicializar el array b con
     * los valores de Bernstein
     *
     */
    private static void iniciaBernstein() {
        GeneraVHDL.b = new double[4][4];
        b[0][0] = 1;
        b[0][1] = 0;
        b[0][2] = 0;
        b[0][3] = 0;
        b[1][0] = 0.5;
        b[1][1] = 0.5;
        b[1][2] = 0;
        b[1][3] = 0;
        b[2][0] = 0.25;
        b[2][1] = 0.5;
        b[2][2] = 0.25;
        b[2][3] = 0;
        b[3][0] = 0.125;
        b[3][1] = 0.375;
        b[3][2] = 0.375;
        b[3][3] = 0.125;
    }

    /**
     * Paso de double a binario
     *
     * @param b:
     *            número en double
```

```
     * @return: número en binario
     */
    private static String doubleToString(double b) {
        String retorno = "";
        long a = 0x1;
        a = a << 32;
        long result = (long) (((float) a) * b);
        retorno = Long.toBinaryString(result);
        while (retorno.length() < 32)
            retorno = "0" + retorno;
        retorno = "\"" + retorno + "\"";
        return retorno;
    }

}
```

Las clases *Desplazamiento* y *Suma* son similares a las del apartado anterior.

Apéndice D

Listado de código VHDL

El presente apéndice contiene el listado de código en VHDL que contiene la arquitectura desarrollada.

D.1. *Hardware sin optimizar*

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity divisor_mallas is
port (
clk : in std_logic;
rst : in std_logic;
entradaDatos : in std_logic_vector(31 downto 0);
salidaDatos : out std_logic_vector(31 downto 0);
entradaControl : in std_logic_vector(31 downto 0);
salidaControl : out std_logic_vector(31 downto 0)
);

end divisor_mallas;

architecture imp of divisor_mallas is

--Señales para los procesos del protocolo
signal datoOk, set, get, leido, escrito, rdy:std_logic;
```

```
signal datosIn, datosOut :
                std_logic_vector(31 downto 0);
type estados  is (espera,direccionando,
                  escribiendo,leyendo);
signal estado: estados;
signal contador: natural;
signal eDecodificador: std_logic;
signal eMultiplexor: std_logic;

--Señales para conectar el divisor con el protocolo
-- (datos de entrada)
signal regE00, regE01, regE02, regE03, regE10, regE11,
       regE12, regE13, regE20, regE21, regE22, regE23,
       regE30, regE31, regE32, regE33 :
       std_logic_vector(31 downto 0);

--Señales para conectar el divisor con el protocolo
-- (datos de salida)
signal regS00, regS01, regS02, regS03, regS04, regS05,
regS06, regS07, regS08, regS09, regS010, regS011,
regS012, regS013, regS014, regS015, regS10,regS11,
regS12, regS13, regS14, regS15, regS16, regS17,
regS18, regS19, regS110, regS111, regS112, regS113,
regS114, regS115, regS20, regS21, regS22, regS23,
regS24, regS25, regS26, regS27, regS28, regS29, regS210,
regS211, regS212, regS213, regS214, regS215, regS30,
regS31, regS32, regS33, regS34, regS35, regS36, regS37,
regS38, regS39, regS310, regS311, regS312, regS313,
regS314, regS315 : std_logic_vector(31 downto 0);

--Señales para los operandos fuente del divisor
signal punto00, punto01, punto02, punto03, punto10,
        punto11, punto12, punto13,
       punto20, punto21, punto22, punto23, punto30,
        punto31, punto32,
       punto33 : std_logic_vector(31 downto 0);

--Señales para los operandos resultado del divisor.
signal malla0N00, malla0N01, malla0N02, malla0N03,
       malla0N10, malla0N11, malla0N12, malla0N13,
       malla0N20, malla0N21, malla0N22, malla0N23,
```

```
        malla0N30, malla0N31, malla0N32, malla0N33,
        malla1N00, malla1N01, malla1N02, malla1N03,
        malla1N10, malla1N11, malla1N12, malla1N13,
        malla1N20, malla1N21, malla1N22, malla1N23,
        malla1N30, malla1N31, malla1N32, malla1N33,
        malla2N00, malla2N01, malla2N02, malla2N03,
        malla2N10, malla2N11, malla2N12, malla2N13,
        malla2N20, malla2N21, malla2N22, malla2N23,
        malla2N30, malla2N31, malla2N32, malla2N33,
        malla3N00, malla3N01, malla3N02, malla3N03,
        malla3N10, malla3N11, malla3N12, malla3N13,
        malla3N20, malla3N21, malla3N22, malla3N23,
        malla3N30, malla3N31, malla3N32, malla3N33 :
         std_logic_vector(31 downto 0);

-- Señales para los resultados intermedios
signal m0N00sumando00,m0N01sumando00,m0N01sumando10,
       m0N02sumando00,m0N02sumando10,
       m0N02sumando20,m0N03sumando00,m0N03sumando10,
       m0N03sumando20,m0N03sumando30,
       m0N10sumando00,m0N10sumando01,m0N11sumando00,
       m0N11sumando01,m0N11sumando10,
       m0N11sumando11,m0N12sumando00,m0N12sumando01,
       m0N12sumando10,m0N12sumando11,
       m0N12sumando20,m0N12sumando21,m0N13sumando00,
       m0N13sumando01,m0N13sumando10,
       m0N13sumando11,m0N13sumando20,m0N13sumando21,
       m0N13sumando30,m0N13sumando31,
       m0N20sumando00,m0N20sumando01,m0N20sumando02,
       m0N21sumando00,m0N21sumando01,
       m0N21sumando02,m0N21sumando10,m0N21sumando11,
       m0N21sumando12,m0N22sumando00,
       m0N22sumando01,m0N22sumando02,m0N22sumando10,
       m0N22sumando11,m0N22sumando12,
       m0N22sumando20,m0N22sumando21,m0N22sumando22,
       m0N23sumando00,m0N23sumando01,
       m0N23sumando02,m0N23sumando10,m0N23sumando11,
       m0N23sumando12,m0N23sumando20,
       m0N23sumando21,m0N23sumando22,m0N23sumando30,
       m0N23sumando31,m0N23sumando32,
       m0N30sumando00,m0N30sumando01,m0N30sumando02,
```

m0N30sumando03 , m0N31sumando00 ,
m0N31sumando01 , m0N31sumando02 , m0N31sumando03 ,
m0N31sumando10 , m0N31sumando11 ,
m0N31sumando12 , m0N31sumando13 , m0N32sumando00 ,
m0N32sumando01 , m0N32sumando02 ,
m0N32sumando03 , m0N32sumando10 , m0N32sumando11 ,
m0N32sumando12 , m0N32sumando13 ,
m0N32sumando20 , m0N32sumando21 , m0N32sumando22 ,
m0N32sumando23 , m0N33sumando00 ,
m0N33sumando01 , m0N33sumando02 , m0N33sumando03 ,
m0N33sumando10 , m0N33sumando11 ,
m0N33sumando12 , m0N33sumando13 , m0N33sumando20 ,
m0N33sumando21 , m0N33sumando22 ,
m0N33sumando23 , m0N33sumando30 , m0N33sumando31 ,
m0N33sumando32 , m0N33sumando33 ,
m1N00sumando00 , m1N00sumando01 , m1N00sumando02 ,
m1N00sumando03 , m1N01sumando00 ,
m1N01sumando01 , m1N01sumando02 , m1N01sumando03 ,
m1N01sumando10 , m1N01sumando11 ,
m1N01sumando12 , m1N01sumando13 , m1N02sumando00 ,
m1N02sumando01 , m1N02sumando02 ,
m1N02sumando03 , m1N02sumando10 , m1N02sumando11 ,
m1N02sumando12 , m1N02sumando13 ,
m1N02sumando20 , m1N02sumando21 , m1N02sumando22 ,
m1N02sumando23 , m1N03sumando00 ,
m1N03sumando01 , m1N03sumando02 , m1N03sumando03 ,
m1N03sumando10 , m1N03sumando11 ,
m1N03sumando12 , m1N03sumando13 , m1N03sumando20 ,
m1N03sumando21 , m1N03sumando22 ,
m1N03sumando23 , m1N03sumando30 , m1N03sumando31 ,
m1N03sumando32 , m1N03sumando33 ,
m1N10sumando00 , m1N10sumando01 , m1N10sumando02 ,
m1N11sumando00 , m1N11sumando01 ,
m1N11sumando02 , m1N11sumando10 , m1N11sumando11 ,
m1N11sumando12 , m1N12sumando00 ,
m1N12sumando01 , m1N12sumando02 , m1N12sumando10 ,
m1N12sumando11 , m1N12sumando12 ,
m1N12sumando20 , m1N12sumando21 , m1N12sumando22 ,
m1N13sumando00 , m1N13sumando01 ,
m1N13sumando02 , m1N13sumando10 , m1N13sumando11 ,
m1N13sumando12 , m1N13sumando20 ,

m1N13sumando21, m1N13sumando22, m1N13sumando30,
m1N13sumando31, m1N13sumando32,
m1N20sumando00, m1N20sumando01, m1N21sumando00,
m1N21sumando01, m1N21sumando10,
m1N21sumando11, m1N22sumando00, m1N22sumando01,
m1N22sumando10, m1N22sumando11,
m1N22sumando20, m1N22sumando21, m1N23sumando00,
m1N23sumando01, m1N23sumando10,
m1N23sumando11, m1N23sumando20, m1N23sumando21,
m1N23sumando30, m1N23sumando31,
m1N30sumando00, m1N31sumando00, m1N31sumando10,
m1N32sumando00, m1N32sumando10,
m1N32sumando20, m1N33sumando00, m1N33sumando10,
m1N33sumando20, m1N33sumando30,
m2N00sumando00, m2N00sumando10, m2N00sumando20,
m2N00sumando30, m2N01sumando00,
m2N01sumando10, m2N01sumando20, m2N02sumando00,
m2N02sumando10, m2N03sumando00,
m2N10sumando00, m2N10sumando01, m2N10sumando10,
m2N10sumando11, m2N10sumando20,
m2N10sumando21, m2N10sumando30, m2N10sumando31,
m2N11sumando00, m2N11sumando01,
m2N11sumando10, m2N11sumando11, m2N11sumando20,
m2N11sumando21, m2N12sumando00,
m2N12sumando01, m2N12sumando10, m2N12sumando11,
m2N13sumando00, m2N13sumando01,
m2N20sumando00, m2N20sumando01, m2N20sumando02,
m2N20sumando10, m2N20sumando11,
m2N20sumando12, m2N20sumando20, m2N20sumando21,
m2N20sumando22, m2N20sumando30,
m2N20sumando31, m2N20sumando32, m2N21sumando00,
m2N21sumando01, m2N21sumando02,
m2N21sumando10, m2N21sumando11, m2N21sumando12,
m2N21sumando20, m2N21sumando21,
m2N21sumando22, m2N22sumando00, m2N22sumando01,
m2N22sumando02, m2N22sumando10,
m2N22sumando11, m2N22sumando12, m2N23sumando00,
m2N23sumando01, m2N23sumando02,
m2N30sumando00, m2N30sumando01, m2N30sumando02,
m2N30sumando03, m2N30sumando10,
m2N30sumando11, m2N30sumando12, m2N30sumando13,

m2N30sumando20 , m2N30sumando21 ,
m2N30sumando22 , m2N30sumando23 , m2N30sumando30 ,
m2N30sumando31 , m2N30sumando32 ,
m2N30sumando33 , m2N31sumando00 , m2N31sumando01 ,
m2N31sumando02 , m2N31sumando03 ,
m2N31sumando10 , m2N31sumando11 , m2N31sumando12 ,
m2N31sumando13 , m2N31sumando20 ,
m2N31sumando21 , m2N31sumando22 , m2N31sumando23 ,
m2N32sumando00 , m2N32sumando01 ,
m2N32sumando02 , m2N32sumando03 , m2N32sumando10 ,
m2N32sumando11 , m2N32sumando12 ,
m2N32sumando13 , m2N33sumando00 , m2N33sumando01 ,
m2N33sumando02 , m2N33sumando03 ,
m3N00sumando00 , m3N00sumando01 , m3N00sumando02 ,
m3N00sumando03 , m3N00sumando10 ,
m3N00sumando11 , m3N00sumando12 , m3N00sumando13 ,
m3N00sumando20 , m3N00sumando21 ,
m3N00sumando22 , m3N00sumando23 , m3N00sumando30 ,
m3N00sumando31 , m3N00sumando32 ,
m3N00sumando33 , m3N01sumando00 , m3N01sumando01 ,
m3N01sumando02 , m3N01sumando03 ,
m3N01sumando10 , m3N01sumando11 , m3N01sumando12 ,
m3N01sumando13 , m3N01sumando20 ,
m3N01sumando21 , m3N01sumando22 , m3N01sumando23 ,
m3N02sumando00 , m3N02sumando01 ,
m3N02sumando02 , m3N02sumando03 , m3N02sumando10 ,
m3N02sumando11 , m3N02sumando12 ,
m3N02sumando13 , m3N03sumando00 , m3N03sumando01 ,
m3N03sumando02 , m3N03sumando03 ,
m3N10sumando00 , m3N10sumando01 , m3N10sumando02 ,
m3N10sumando10 , m3N10sumando11 ,
m3N10sumando12 , m3N10sumando20 , m3N10sumando21 ,
m3N10sumando22 , m3N10sumando30 ,
m3N10sumando31 , m3N10sumando32 , m3N11sumando00 ,
m3N11sumando01 , m3N11sumando02 ,
m3N11sumando10 , m3N11sumando11 , m3N11sumando12 ,
m3N11sumando20 , m3N11sumando21 ,
m3N11sumando22 , m3N12sumando00 , m3N12sumando01 ,
m3N12sumando02 , m3N12sumando10 ,
m3N12sumando11 , m3N12sumando12 , m3N13sumando00 ,
m3N13sumando01 , m3N13sumando02 ,

```
        m3N20sumando00, m3N20sumando01, m3N20sumando10,
        m3N20sumando11, m3N20sumando20,
        m3N20sumando21, m3N20sumando30, m3N20sumando31,
        m3N21sumando00, m3N21sumando01,
        m3N21sumando10, m3N21sumando11, m3N21sumando20,
        m3N21sumando21, m3N22sumando00,
        m3N22sumando01, m3N22sumando10, m3N22sumando11,
        m3N23sumando00, m3N23sumando01,
        m3N30sumando00, m3N30sumando10, m3N30sumando20,
        m3N30sumando30, m3N31sumando00,
        m3N31sumando10, m3N31sumando20, m3N32sumando00,
        m3N32sumando10, m3N33sumando00:
std_logic_vector (31 downto 0);

begin

-------------------------------
 ------- PROTOCOLO -------
-------------------------------
        -- Entradas de control:
        set <= entradaControl(17);
        get <= entradaControl(18);
        leido <= entradaControl(19);
        escrito <= entradaControl(20);

        -- Salidas de control:
        salidaControl(0) <= datoOk;
        salidaControl(1) <= rdy;
        salidaControl(31 downto 2) <= (others => '0');

        -- Datos
        salidaDatos <= datosOut;
        datosIn <= entradaDatos;

        controlador:
        process(clk, rst)
        begin
                rdy <= '0';
                datoOk <= '0';
                eDecodificador <= '0';
                eMultiplexor <= '0';
```

```
if (rst = '0') then
        estado <= espera;
        contador <= 0;
elsif (clk'event and clk = '1') then
        case estado is
          when espera =>
            contador <= 0;
            if(set = '1') then
              estado <= escribiendo;
            elsif(get = '1') then
                estado <= leyendo;
            end if;

          when escribiendo =>
            if(escrito = '1')then
             if(contador = 15) then
               estado <= espera;
             else
               estado <= direccionando;
              contador <= contador + 1;
             end if;
           end if;

          when direccionando =>
            if(set='1') then
                estado<=escribiendo;
            elsif(get='1') then
                estado<=leyendo;
            end if;

          when leyendo =>
            if(leido = '1') then
             if(contador = 63) then
               estado <= espera;
             else
               estado <= direccionando;
              contador <= contador + 1;
             end if;
           end if;

          when others =>
```

```
                        null;
                end case;
        end if;

        case estado is
                when espera =>
                        rdy <= '1';
                when escribiendo =>
                        eDecodificador <= '1';
                        datoOk <= '1';
                when leyendo =>
                        eMultiplexor <= '1';
                        datoOk <= '1';
                when direccionando =>
                        null;
                when others =>
                        null;
        end case;
end process controlador;

Decodificador:
process (clk,rst)
begin
        if(rst='0') then
                regE00 <= (others => '0');
                regE01 <= (others => '0');
                regE02 <= (others => '0');
                regE03 <= (others => '0');
                regE10 <= (others => '0');
                regE11 <= (others => '0');
                regE12 <= (others => '0');
                regE13 <= (others => '0');
                regE20 <= (others => '0');
                regE21 <= (others => '0');
                regE22 <= (others => '0');
                regE23 <= (others => '0');
                regE30 <= (others => '0');
                regE31 <= (others => '0');
                regE32 <= (others => '0');
                regE33 <= (others => '0');
        elsif(clk'event and clk='1') then
```

```
                if(eDecodificador = '1') then
                  case contador is
                    when 0 => regE00 <= datosIn;
                    when 1 => regE01 <= datosIn;
                    when 2 => regE02 <= datosIn;
                    when 3 => regE03 <= datosIn;
                    when 4 => regE10 <= datosIn;
                    when 5 => regE11 <= datosIn;
                    when 6 => regE12 <= datosIn;
                    when 7 => regE13 <= datosIn;
                    when 8 => regE20 <= datosIn;
                    when 9 => regE21 <= datosIn;
                    when 10 => regE22 <= datosIn;
                    when 11 => regE23 <= datosIn;
                    when 12 => regE30 <= datosIn;
                    when 13 => regE31 <= datosIn;
                    when 14 => regE32 <= datosIn;
                    when 15 => regE33 <= datosIn;
                    when others =>null;
                  end case;
                end if;
        end if;
end process;

multiplexor:
process(rst,clk)
begin
        datosOut <= (others => '0');
        if(eMultiplexor='1') then
           case contador is
                when 0 => datosOut <= regS00;
                when 1 => datosOut <= regS01;
                when 2 => datosOut <= regS02;
                when 3 => datosOut <= regS03;
                when 4 => datosOut <= regS04;
                when 5 => datosOut <= regS05;
                when 6 => datosOut <= regS06;
                when 7 => datosOut <= regS07;
                when 8 => datosOut <= regS08;
                when 9 => datosOut <= regS09;
                when 10 => datosOut <= regS010;
```

```
when 11 => datosOut <= regS011;
when 12 => datosOut <= regS012;
when 13 => datosOut <= regS013;
when 14 => datosOut <= regS014;
when 15 => datosOut <= regS015;
when 16 => datosOut <= regS10;
when 17 => datosOut <= regS11;
when 18 => datosOut <= regS12;
when 19 => datosOut <= regS13;
when 20 => datosOut <= regS14;
when 21 => datosOut <= regS15;
when 22 => datosOut <= regS16;
when 23 => datosOut <= regS17;
when 24 => datosOut <= regS18;
when 25 => datosOut <= regS19;
when 26 => datosOut <= regS110;
when 27 => datosOut <= regS111;
when 28 => datosOut <= regS112;
when 29 => datosOut <= regS113;
when 30 => datosOut <= regS114;
when 31 => datosOut <= regS115;
when 32 => datosOut <= regS20;
when 33 => datosOut <= regS21;
when 34 => datosOut <= regS22;
when 35 => datosOut <= regS23;
when 36 => datosOut <= regS24;
when 37 => datosOut <= regS25;
when 38 => datosOut <= regS26;
when 39 => datosOut <= regS27;
when 40 => datosOut <= regS28;
when 41 => datosOut <= regS29;
when 42 => datosOut <= regS210;
when 43 => datosOut <= regS211;
when 44 => datosOut <= regS212;
when 45 => datosOut <= regS213;
when 46 => datosOut <= regS214;
when 47 => datosOut <= regS215;
when 48 => datosOut <= regS30;
when 49 => datosOut <= regS31;
when 50 => datosOut <= regS32;
when 51 => datosOut <= regS33;
```

```
                when 52 => datosOut <= regS34;
                when 53 => datosOut <= regS35;
                when 54 => datosOut <= regS36;
                when 55 => datosOut <= regS37;
                when 56 => datosOut <= regS38;
                when 57 => datosOut <= regS39;
                when 58 => datosOut <= regS310;
                when 59 => datosOut <= regS311;
                when 60 => datosOut <= regS312;
                when 61 => datosOut <= regS313;
                when 62 => datosOut <= regS314;
                when 63 => datosOut <= regS315;
                when others =>
                    datosOut <= (others => '0');
            end case;
        end if;
    end process;

-------------------------
------- DIVISOR -------
-------------------------

        ------------------------  Malla 0--------------------
m0N00sumando00 <=  punto00;
malla0N00 <= m0N00sumando00;

m0N01sumando00 <= "0" & punto00(31 downto 1);
m0N01sumando10 <= "0" & punto10(31 downto 1);
malla0N01 <= m0N01sumando00 + m0N01sumando10;

m0N02sumando00 <= "00" & punto00(31 downto 2);
m0N02sumando10 <= "0" & punto10(31 downto 1);
m0N02sumando20 <= "00" & punto20(31 downto 2);
malla0N02 <= m0N02sumando00 + m0N02sumando10 +
m0N02sumando20;

m0N03sumando00 <= "000" & punto00(31 downto 3);
m0N03sumando10 <=( "00" & punto10(31 downto 2))+
("000" & punto10(31 downto 3));
m0N03sumando20 <=( "00" & punto20(31 downto 2))+
("000" & punto20(31 downto 3));
```

```
m0N03sumando30 <= "000" & punto30(31 downto 3);
malla0N03 <= m0N03sumando00 + m0N03sumando10 +
m0N03sumando20 + m0N03sumando30;

m0N10sumando00 <= "0" & punto00(31 downto 1);
m0N10sumando01 <= "0" & punto01(31 downto 1);
malla0N10 <= m0N10sumando00 + m0N10sumando01;

m0N11sumando00 <= "00" & punto00(31 downto 2);
m0N11sumando01 <= "00" & punto01(31 downto 2);
m0N11sumando10 <= "00" & punto10(31 downto 2);
m0N11sumando11 <= "00" & punto11(31 downto 2);
malla0N11 <= m0N11sumando00 + m0N11sumando01 +
m0N11sumando10 + m0N11sumando11;

m0N12sumando00 <= "000" & punto00(31 downto 3);
m0N12sumando01 <= "000" & punto01(31 downto 3);
m0N12sumando10 <= "00" & punto10(31 downto 2);
m0N12sumando11 <= "00" & punto11(31 downto 2);
m0N12sumando20 <= "000" & punto20(31 downto 3);
m0N12sumando21 <= "000" & punto21(31 downto 3);
malla0N12 <= m0N12sumando00 + m0N12sumando01 +
m0N12sumando10 + m0N12sumando11 + m0N12sumando20 +
 m0N12sumando21;

m0N13sumando00 <= "0000" & punto00(31 downto 4);
m0N13sumando01 <= "0000" & punto01(31 downto 4);
m0N13sumando10 <=( "000" & punto10(31 downto 3))+
 ("0000" & punto10(31 downto 4));
m0N13sumando11 <=( "000" & punto11(31 downto 3))+
 ("0000" & punto11(31 downto 4));
m0N13sumando20 <=( "000" & punto20(31 downto 3))+
 ("0000" & punto20(31 downto 4));
m0N13sumando21 <=( "000" & punto21(31 downto 3))+
 ("0000" & punto21(31 downto 4));
m0N13sumando30 <= "0000" & punto30(31 downto 4);
m0N13sumando31 <= "0000" & punto31(31 downto 4);
malla0N13 <= m0N13sumando00 + m0N13sumando01 +
m0N13sumando10 + m0N13sumando11 + m0N13sumando20 +
m0N13sumando21 + m0N13sumando30 + m0N13sumando31;
```

```
m0N20sumando00 <= "00" & punto00(31 downto 2);
m0N20sumando01 <= "0" & punto01(31 downto 1);
m0N20sumando02 <= "00" & punto02(31 downto 2);
malla0N20 <= m0N20sumando00 + m0N20sumando01 +
m0N20sumando02;

m0N21sumando00 <= "000" & punto00(31 downto 3);
m0N21sumando01 <= "00" & punto01(31 downto 2);
m0N21sumando02 <= "000" & punto02(31 downto 3);
m0N21sumando10 <= "000" & punto10(31 downto 3);
m0N21sumando11 <= "00" & punto11(31 downto 2);
m0N21sumando12 <= "000" & punto12(31 downto 3);
malla0N21 <= m0N21sumando00 + m0N21sumando01 +
m0N21sumando02 + m0N21sumando10 + m0N21sumando11 +
m0N21sumando12;

m0N22sumando00 <= "0000" & punto00(31 downto 4);
m0N22sumando01 <= "000" & punto01(31 downto 3);
m0N22sumando02 <= "0000" & punto02(31 downto 4);
m0N22sumando10 <= "000" & punto10(31 downto 3);
m0N22sumando11 <= "00" & punto11(31 downto 2);
m0N22sumando12 <= "000" & punto12(31 downto 3);
m0N22sumando20 <= "0000" & punto20(31 downto 4);
m0N22sumando21 <= "000" & punto21(31 downto 3);
m0N22sumando22 <= "0000" & punto22(31 downto 4);
malla0N22 <= m0N22sumando00 + m0N22sumando01 +
m0N22sumando02 + m0N22sumando10 + m0N22sumando11 +
m0N22sumando12 + m0N22sumando20 + m0N22sumando21 +
m0N22sumando22;

m0N23sumando00 <= "00000" & punto00(31 downto 5);
m0N23sumando01 <= "0000" & punto01(31 downto 4);
m0N23sumando02 <= "00000" & punto02(31 downto 5);
m0N23sumando10 <=( "0000" & punto10(31 downto 4))+
("00000" & punto10(31 downto 5));
m0N23sumando11 <=( "000" & punto11(31 downto 3))+
("0000" & punto11(31 downto 4));
m0N23sumando12 <=( "0000" & punto12(31 downto 4))+
("00000" & punto12(31 downto 5));
m0N23sumando20 <=( "0000" & punto20(31 downto 4))+
("00000" & punto20(31 downto 5));
```

```
m0N23sumando21 <=( "000" & punto21(31 downto 3))+
("0000" & punto21(31 downto 4));
m0N23sumando22 <=( "0000" & punto22(31 downto 4))+
("00000" & punto22(31 downto 5));
m0N23sumando30 <= "00000" & punto30(31 downto 5);
m0N23sumando31 <= "0000" & punto31(31 downto 4);
m0N23sumando32 <= "00000" & punto32(31 downto 5);
malla0N23 <= m0N23sumando00 + m0N23sumando01 +
m0N23sumando02 + m0N23sumando10 + m0N23sumando11 +
m0N23sumando12 + m0N23sumando20 + m0N23sumando21 +
m0N23sumando22 + m0N23sumando30 + m0N23sumando31 +
m0N23sumando32;

m0N30sumando00 <= "000" & punto00(31 downto 3);
m0N30sumando01 <=( "00" & punto01(31 downto 2))+
("000" & punto01(31 downto 3));
m0N30sumando02 <=( "00" & punto02(31 downto 2))+
("000" & punto02(31 downto 3));
m0N30sumando03 <= "000" & punto03(31 downto 3);
malla0N30 <= m0N30sumando00 + m0N30sumando01 +
m0N30sumando02 + m0N30sumando03;

m0N31sumando00 <= "0000" & punto00(31 downto 4);
m0N31sumando01 <=( "000" & punto01(31 downto 3))+
("0000" & punto01(31 downto 4));
m0N31sumando02 <=( "000" & punto02(31 downto 3))+
("0000" & punto02(31 downto 4));
m0N31sumando03 <= "0000" & punto03(31 downto 4);
m0N31sumando10 <= "0000" & punto10(31 downto 4);
m0N31sumando11 <=( "000" & punto11(31 downto 3))+
("0000" & punto11(31 downto 4));
m0N31sumando12 <=( "000" & punto12(31 downto 3))+
("0000" & punto12(31 downto 4));
m0N31sumando13 <= "0000" & punto13(31 downto 4);
malla0N31 <= m0N31sumando00 + m0N31sumando01 +
m0N31sumando02 + m0N31sumando03 + m0N31sumando10 +
m0N31sumando11 + m0N31sumando12 + m0N31sumando13;

m0N32sumando00 <= "00000" & punto00(31 downto 5);
m0N32sumando01 <=( "0000" & punto01(31 downto 4))+
("00000" & punto01(31 downto 5));
```

```
m0N32sumando02 <=( "0000" & punto02(31 downto 4))+
("00000" & punto02(31 downto 5));
m0N32sumando03 <= "00000" & punto03(31 downto 5);
m0N32sumando10 <= "0000" & punto10(31 downto 4);
m0N32sumando11 <=( "000" & punto11(31 downto 3))+
("0000" & punto11(31 downto 4));
m0N32sumando12 <=( "000" & punto12(31 downto 3))+
("0000" & punto12(31 downto 4));
m0N32sumando13 <= "0000" & punto13(31 downto 4);
m0N32sumando20 <= "00000" & punto20(31 downto 5);
m0N32sumando21 <=( "0000" & punto21(31 downto 4))+
("00000" & punto21(31 downto 5));
m0N32sumando22 <=( "0000" & punto22(31 downto 4))+
("00000" & punto22(31 downto 5));
m0N32sumando23 <= "00000" & punto23(31 downto 5);
malla0N32 <= m0N32sumando00 + m0N32sumando01 +
m0N32sumando02 + m0N32sumando03 + m0N32sumando10 +
m0N32sumando11 + m0N32sumando12 + m0N32sumando13 +
m0N32sumando20 + m0N32sumando21 + m0N32sumando22 +
m0N32sumando23;

m0N33sumando00 <= "000000" & punto00(31 downto 6);
m0N33sumando01 <=( "00000" & punto01(31 downto 5))+
("000000" & punto01(31 downto 6));
m0N33sumando02 <=( "00000" & punto02(31 downto 5))+
("000000" & punto02(31 downto 6));
m0N33sumando03 <= "000000" & punto03(31 downto 6);
m0N33sumando10 <=( "00000" & punto10(31 downto 5))+
("000000" & punto10(31 downto 6));
m0N33sumando11 <=( "000" & punto11(31 downto 3))+
("000000" & punto11(31 downto 6));
m0N33sumando12 <=( "000" & punto12(31 downto 3))+
("000000" & punto12(31 downto 6));
m0N33sumando13 <=( "00000" & punto13(31 downto 5))+
("000000" & punto13(31 downto 6));
m0N33sumando20 <=( "00000" & punto20(31 downto 5))+
("000000" & punto20(31 downto 6));
m0N33sumando21 <=( "000" & punto21(31 downto 3))+
("000000" & punto21(31 downto 6));
m0N33sumando22 <=( "000" & punto22(31 downto 3))+
("000000" & punto22(31 downto 6));
```

```
m0N33sumando23 <=( "00000" & punto23(31 downto 5))+
("000000" & punto23(31 downto 6));
m0N33sumando30 <= "000000" & punto30(31 downto 6);
m0N33sumando31 <=( "00000" & punto31(31 downto 5))+
("000000" & punto31(31 downto 6));
m0N33sumando32 <=( "00000" & punto32(31 downto 5))+
("000000" & punto32(31 downto 6));
m0N33sumando33 <= "000000" & punto33(31 downto 6);
malla0N33 <= m0N33sumando00 + m0N33sumando01 +
m0N33sumando02 + m0N33sumando03 + m0N33sumando10 +
m0N33sumando11 + m0N33sumando12 + m0N33sumando13 +
m0N33sumando20 + m0N33sumando21 + m0N33sumando22 +
m0N33sumando23 + m0N33sumando30 + m0N33sumando31 +
m0N33sumando32 + m0N33sumando33;
```

——————— *Malla 1* ———————

```
m1N00sumando00 <= "000" & punto00(31 downto 3);
m1N00sumando01 <=( "00" & punto01(31 downto 2))+
("000" & punto01(31 downto 3));
m1N00sumando02 <=( "00" & punto02(31 downto 2))+
("000" & punto02(31 downto 3));
m1N00sumando03 <= "000" & punto03(31 downto 3);
malla1N00 <= m1N00sumando00 + m1N00sumando01 +
m1N00sumando02 + m1N00sumando03;

m1N01sumando00 <= "0000" & punto00(31 downto 4);
m1N01sumando01 <=( "000" & punto01(31 downto 3))+
("0000" & punto01(31 downto 4));
m1N01sumando02 <=( "000" & punto02(31 downto 3))+
("0000" & punto02(31 downto 4));
m1N01sumando03 <= "0000" & punto03(31 downto 4);
m1N01sumando10 <= "0000" & punto10(31 downto 4);
m1N01sumando11 <=( "000" & punto11(31 downto 3))+
("0000" & punto11(31 downto 4));
m1N01sumando12 <=( "000" & punto12(31 downto 3))+
("0000" & punto12(31 downto 4));
m1N01sumando13 <= "0000" & punto13(31 downto 4);
malla1N01 <= m1N01sumando00 + m1N01sumando01 +
m1N01sumando02 + m1N01sumando03 + m1N01sumando10 +
m1N01sumando11 + m1N01sumando12 + m1N01sumando13;
```

```
m1N02sumando00 <= "00000" & punto00(31 downto 5);
m1N02sumando01 <=( "0000" & punto01(31 downto 4))+
("00000" & punto01(31 downto 5));
m1N02sumando02 <=( "0000" & punto02(31 downto 4))+
("00000" & punto02(31 downto 5));
m1N02sumando03 <= "00000" & punto03(31 downto 5);
m1N02sumando10 <= "0000" & punto10(31 downto 4);
m1N02sumando11 <=( "000" & punto11(31 downto 3))+
("0000" & punto11(31 downto 4));
m1N02sumando12 <=( "000" & punto12(31 downto 3))+
("0000" & punto12(31 downto 4));
m1N02sumando13 <= "0000" & punto13(31 downto 4);
m1N02sumando20 <= "00000" & punto20(31 downto 5);
m1N02sumando21 <=( "0000" & punto21(31 downto 4))+
("00000" & punto21(31 downto 5));
m1N02sumando22 <=( "0000" & punto22(31 downto 4))+
("00000" & punto22(31 downto 5));
m1N02sumando23 <= "00000" & punto23(31 downto 5);
malla1N02 <= m1N02sumando00 + m1N02sumando01 +
 m1N02sumando02 + m1N02sumando03 + m1N02sumando10 +
m1N02sumando11 + m1N02sumando12 + m1N02sumando13 +
m1N02sumando20 + m1N02sumando21 + m1N02sumando22 +
m1N02sumando23;

m1N03sumando00 <= "000000" & punto00(31 downto 6);
m1N03sumando01 <=( "00000" & punto01(31 downto 5))+
("000000" & punto01(31 downto 6));
m1N03sumando02 <=( "00000" & punto02(31 downto 5))+
("000000" & punto02(31 downto 6));
m1N03sumando03 <= "000000" & punto03(31 downto 6);
m1N03sumando10 <=( "00000" & punto10(31 downto 5))+
("000000" & punto10(31 downto 6));
m1N03sumando11 <=( "000" & punto11(31 downto 3))+
("000000" & punto11(31 downto 6));
m1N03sumando12 <=( "000" & punto12(31 downto 3))+
("000000" & punto12(31 downto 6));
m1N03sumando13 <=( "00000" & punto13(31 downto 5))+
("000000" & punto13(31 downto 6));
m1N03sumando20 <=( "00000" & punto20(31 downto 5))+
("000000" & punto20(31 downto 6));
m1N03sumando21 <=( "000" & punto21(31 downto 3))+
```

```
("000000" & punto21(31 downto 6));
m1N03sumando22 <=( "000" & punto22(31 downto 3))+
("000000" & punto22(31 downto 6));
m1N03sumando23 <=( "00000" & punto23(31 downto 5))+
("000000" & punto23(31 downto 6));
m1N03sumando30 <= "000000" & punto30(31 downto 6);
m1N03sumando31 <=( "00000" & punto31(31 downto 5))+
("000000" & punto31(31 downto 6));
m1N03sumando32 <=( "00000" & punto32(31 downto 5))+
("000000" & punto32(31 downto 6));
m1N03sumando33 <= "000000" & punto33(31 downto 6);
malla1N03 <= m1N03sumando00 + m1N03sumando01 +
m1N03sumando02 + m1N03sumando03 + m1N03sumando10 +
m1N03sumando11 + m1N03sumando12 + m1N03sumando13 +
m1N03sumando20 + m1N03sumando21 + m1N03sumando22 +
m1N03sumando23 + m1N03sumando30 + m1N03sumando31 +
m1N03sumando32 + m1N03sumando33;

m1N10sumando00 <= "00" & punto01(31 downto 2);
m1N10sumando01 <= "0" & punto02(31 downto 1);
m1N10sumando02 <= "00" & punto03(31 downto 2);
malla1N10 <= m1N10sumando00 + m1N10sumando01 +
m1N10sumando02;

m1N11sumando00 <= "000" & punto01(31 downto 3);
m1N11sumando01 <= "00" & punto02(31 downto 2);
m1N11sumando02 <= "000" & punto03(31 downto 3);
m1N11sumando10 <= "000" & punto11(31 downto 3);
m1N11sumando11 <= "00" & punto12(31 downto 2);
m1N11sumando12 <= "000" & punto13(31 downto 3);
malla1N11 <= m1N11sumando00 + m1N11sumando01 +
m1N11sumando02 + m1N11sumando10 + m1N11sumando11 +
m1N11sumando12;

m1N12sumando00 <= "0000" & punto01(31 downto 4);
m1N12sumando01 <= "000" & punto02(31 downto 3);
m1N12sumando02 <= "0000" & punto03(31 downto 4);
m1N12sumando10 <= "000" & punto11(31 downto 3);
m1N12sumando11 <= "00" & punto12(31 downto 2);
m1N12sumando12 <= "000" & punto13(31 downto 3);
m1N12sumando20 <= "0000" & punto21(31 downto 4);
```

```
m1N12sumando21 <= "000" & punto22(31 downto 3);
m1N12sumando22 <= "0000" & punto23(31 downto 4);
malla1N12 <= m1N12sumando00 + m1N12sumando01 +
m1N12sumando02 + m1N12sumando10 + m1N12sumando11 +
m1N12sumando12 + m1N12sumando20 + m1N12sumando21 +
m1N12sumando22;

m1N13sumando00 <= "00000" & punto01(31 downto 5);
m1N13sumando01 <= "0000" & punto02(31 downto 4);
m1N13sumando02 <= "00000" & punto03(31 downto 5);
m1N13sumando10 <=( "0000" & punto11(31 downto 4))+
("00000" & punto11(31 downto 5));
m1N13sumando11 <=( "000" & punto12(31 downto 3))+
("0000" & punto12(31 downto 4));
m1N13sumando12 <=( "0000" & punto13(31 downto 4))+
("00000" & punto13(31 downto 5));
m1N13sumando20 <=( "0000" & punto21(31 downto 4))+
("00000" & punto21(31 downto 5));
m1N13sumando21 <=( "000" & punto22(31 downto 3))+
("0000" & punto22(31 downto 4));
m1N13sumando22 <=( "0000" & punto23(31 downto 4))+
("00000" & punto23(31 downto 5));
m1N13sumando30 <= "00000" & punto31(31 downto 5);
m1N13sumando31 <= "0000" & punto32(31 downto 4);
m1N13sumando32 <= "00000" & punto33(31 downto 5);
malla1N13 <= m1N13sumando00 + m1N13sumando01 +
m1N13sumando02 + m1N13sumando10 + m1N13sumando11 +
m1N13sumando12 + m1N13sumando20 + m1N13sumando21 +
m1N13sumando22 + m1N13sumando30 + m1N13sumando31 +
m1N13sumando32;

m1N20sumando00 <= "0" & punto02(31 downto 1);
m1N20sumando01 <= "0" & punto03(31 downto 1);
malla1N20 <= m1N20sumando00 + m1N20sumando01;

m1N21sumando00 <= "00" & punto02(31 downto 2);
m1N21sumando01 <= "00" & punto03(31 downto 2);
m1N21sumando10 <= "00" & punto12(31 downto 2);
m1N21sumando11 <= "00" & punto13(31 downto 2);
malla1N21 <= m1N21sumando00 + m1N21sumando01 +
m1N21sumando10 + m1N21sumando11;
```

```
m1N22sumando00 <= "000" & punto02(31 downto 3);
m1N22sumando01 <= "000" & punto03(31 downto 3);
m1N22sumando10 <= "00" & punto12(31 downto 2);
m1N22sumando11 <= "00" & punto13(31 downto 2);
m1N22sumando20 <= "000" & punto22(31 downto 3);
m1N22sumando21 <= "000" & punto23(31 downto 3);
malla1N22 <= m1N22sumando00 + m1N22sumando01 +
m1N22sumando10 + m1N22sumando11 + m1N22sumando20 +
m1N22sumando21;

m1N23sumando00 <= "0000" & punto02(31 downto 4);
m1N23sumando01 <= "0000" & punto03(31 downto 4);
m1N23sumando10 <=( "000" & punto12(31 downto 3))+
("0000" & punto12(31 downto 4));
m1N23sumando11 <=( "000" & punto13(31 downto 3))+
("0000" & punto13(31 downto 4));
m1N23sumando20 <=( "000" & punto22(31 downto 3))+
("0000" & punto22(31 downto 4));
m1N23sumando21 <=( "000" & punto23(31 downto 3))+
("0000" & punto23(31 downto 4));
m1N23sumando30 <= "0000" & punto32(31 downto 4);
m1N23sumando31 <= "0000" & punto33(31 downto 4);
malla1N23 <= m1N23sumando00 + m1N23sumando01 +
m1N23sumando10 + m1N23sumando11 + m1N23sumando20 +
m1N23sumando21 + m1N23sumando30 + m1N23sumando31;

m1N30sumando00 <=  punto03;
malla1N30 <= m1N30sumando00;

m1N31sumando00 <= "0" & punto03(31 downto 1);
m1N31sumando10 <= "0" & punto13(31 downto 1);
malla1N31 <= m1N31sumando00 + m1N31sumando10;

m1N32sumando00 <= "00" & punto03(31 downto 2);
m1N32sumando10 <= "0" & punto13(31 downto 1);
m1N32sumando20 <= "00" & punto23(31 downto 2);
malla1N32 <= m1N32sumando00 + m1N32sumando10 +
m1N32sumando20;

m1N33sumando00 <= "000" & punto03(31 downto 3);
```

```
m1N33sumando10 <=( "00" & punto13(31 downto 2))+
("000" & punto13(31 downto 3));
m1N33sumando20 <=( "00" & punto23(31 downto 2))+
("000" & punto23(31 downto 3));
m1N33sumando30 <= "000" & punto33(31 downto 3);
malla1N33 <= m1N33sumando00 + m1N33sumando10 +
m1N33sumando20 + m1N33sumando30;

------------------- Malla 2 -------------------
m2N00sumando00 <= "000" & punto00(31 downto 3);
m2N00sumando10 <=( "00" & punto10(31 downto 2))+
("000" & punto10(31 downto 3));
m2N00sumando20 <=( "00" & punto20(31 downto 2))+
("000" & punto20(31 downto 3));
m2N00sumando30 <= "000" & punto30(31 downto 3);
malla2N00 <= m2N00sumando00 + m2N00sumando10 +
m2N00sumando20 + m2N00sumando30;

m2N01sumando00 <= "00" & punto10(31 downto 2);
m2N01sumando10 <= "0" & punto20(31 downto 1);
m2N01sumando20 <= "00" & punto30(31 downto 2);
malla2N01 <= m2N01sumando00 + m2N01sumando10 +
m2N01sumando20;

m2N02sumando00 <= "0" & punto20(31 downto 1);
m2N02sumando10 <= "0" & punto30(31 downto 1);
malla2N02 <= m2N02sumando00 + m2N02sumando10;

m2N03sumando00 <=  punto30;
malla2N03 <= m2N03sumando00;

m2N10sumando00 <= "0000" & punto00(31 downto 4);
m2N10sumando01 <= "0000" & punto01(31 downto 4);
m2N10sumando10 <=( "000" & punto10(31 downto 3))+
("0000" & punto10(31 downto 4));
m2N10sumando11 <=( "000" & punto11(31 downto 3))+
("0000" & punto11(31 downto 4));
m2N10sumando20 <=( "000" & punto20(31 downto 3))+
("0000" & punto20(31 downto 4));
m2N10sumando21 <=( "000" & punto21(31 downto 3))+
("0000" & punto21(31 downto 4));
```

```
m2N10sumando30 <= "0000" & punto30(31 downto 4);
m2N10sumando31 <= "0000" & punto31(31 downto 4);
malla2N10 <= m2N10sumando00 + m2N10sumando01 +
m2N10sumando10 + m2N10sumando11 + m2N10sumando20 +
m2N10sumando21 + m2N10sumando30 + m2N10sumando31;

m2N11sumando00 <= "000" & punto10(31 downto 3);
m2N11sumando01 <= "000" & punto11(31 downto 3);
m2N11sumando10 <= "00" & punto20(31 downto 2);
m2N11sumando11 <= "00" & punto21(31 downto 2);
m2N11sumando20 <= "000" & punto30(31 downto 3);
m2N11sumando21 <= "000" & punto31(31 downto 3);
malla2N11 <= m2N11sumando00 + m2N11sumando01 +
m2N11sumando10 + m2N11sumando11 + m2N11sumando20 +
m2N11sumando21;

m2N12sumando00 <= "00" & punto20(31 downto 2);
m2N12sumando01 <= "00" & punto21(31 downto 2);
m2N12sumando10 <= "00" & punto30(31 downto 2);
m2N12sumando11 <= "00" & punto31(31 downto 2);
malla2N12 <= m2N12sumando00 + m2N12sumando01 +
m2N12sumando10 + m2N12sumando11;

m2N13sumando00 <= "0" & punto30(31 downto 1);
m2N13sumando01 <= "0" & punto31(31 downto 1);
malla2N13 <= m2N13sumando00 + m2N13sumando01;

m2N20sumando00 <= "00000" & punto00(31 downto 5);
m2N20sumando01 <= "0000" & punto01(31 downto 4);
m2N20sumando02 <= "00000" & punto02(31 downto 5);
m2N20sumando10 <=( "0000" & punto10(31 downto 4))+
("00000" & punto10(31 downto 5));
m2N20sumando11 <=( "000" & punto11(31 downto 3))+
("0000" & punto11(31 downto 4));
m2N20sumando12 <=( "0000" & punto12(31 downto 4))+
("00000" & punto12(31 downto 5));
m2N20sumando20 <=( "0000" & punto20(31 downto 4))+
("00000" & punto20(31 downto 5));
m2N20sumando21 <=( "000" & punto21(31 downto 3))+
("0000" & punto21(31 downto 4));
m2N20sumando22 <=( "0000" & punto22(31 downto 4))+
```

```
("00000" & punto22(31 downto 5));
m2N20sumando30 <= "00000" & punto30(31 downto 5);
m2N20sumando31 <= "0000" & punto31(31 downto 4);
m2N20sumando32 <= "00000" & punto32(31 downto 5);
malla2N20 <= m2N20sumando00 + m2N20sumando01 +
m2N20sumando02 + m2N20sumando10 + m2N20sumando11 +
m2N20sumando12 + m2N20sumando20 + m2N20sumando21 +
m2N20sumando22 + m2N20sumando30 + m2N20sumando31 +
m2N20sumando32;

m2N21sumando00 <= "0000" & punto10(31 downto 4);
m2N21sumando01 <= "000" & punto11(31 downto 3);
m2N21sumando02 <= "0000" & punto12(31 downto 4);
m2N21sumando10 <= "000" & punto20(31 downto 3);
m2N21sumando11 <= "00" & punto21(31 downto 2);
m2N21sumando12 <= "000" & punto22(31 downto 3);
m2N21sumando20 <= "0000" & punto30(31 downto 4);
m2N21sumando21 <= "000" & punto31(31 downto 3);
m2N21sumando22 <= "0000" & punto32(31 downto 4);
malla2N21 <= m2N21sumando00 + m2N21sumando01 +
m2N21sumando02 + m2N21sumando10 + m2N21sumando11 +
m2N21sumando12 + m2N21sumando20 + m2N21sumando21 +
m2N21sumando22;

m2N22sumando00 <= "000" & punto20(31 downto 3);
m2N22sumando01 <= "00" & punto21(31 downto 2);
m2N22sumando02 <= "000" & punto22(31 downto 3);
m2N22sumando10 <= "000" & punto30(31 downto 3);
m2N22sumando11 <= "00" & punto31(31 downto 2);
m2N22sumando12 <= "000" & punto32(31 downto 3);
malla2N22 <= m2N22sumando00 + m2N22sumando01 +
m2N22sumando02 + m2N22sumando10 + m2N22sumando11 +
m2N22sumando12;

m2N23sumando00 <= "00" & punto30(31 downto 2);
m2N23sumando01 <= "0" & punto31(31 downto 1);
m2N23sumando02 <= "00" & punto32(31 downto 2);
malla2N23 <= m2N23sumando00 + m2N23sumando01 +
m2N23sumando02;

m2N30sumando00 <= "000000" & punto00(31 downto 6);
```

```
m2N30sumando01 <=( "00000" & punto01(31 downto 5))+
 ("000000" & punto01(31 downto 6));
m2N30sumando02 <=( "00000" & punto02(31 downto 5))+
 ("000000" & punto02(31 downto 6));
m2N30sumando03 <= "000000" & punto03(31 downto 6);
m2N30sumando10 <=( "00000" & punto10(31 downto 5))+
("000000" & punto10(31 downto 6));
m2N30sumando11 <=( "000" & punto11(31 downto 3))+
 ("000000" & punto11(31 downto 6));
m2N30sumando12 <=( "000" & punto12(31 downto 3))+
("000000" & punto12(31 downto 6));
m2N30sumando13 <=( "00000" & punto13(31 downto 5))+
("000000" & punto13(31 downto 6));
m2N30sumando20 <=( "00000" & punto20(31 downto 5))+
("000000" & punto20(31 downto 6));
m2N30sumando21 <=( "000" & punto21(31 downto 3))+
("000000" & punto21(31 downto 6));
m2N30sumando22 <=( "000" & punto22(31 downto 3))+
("000000" & punto22(31 downto 6));
m2N30sumando23 <=( "00000" & punto23(31 downto 5))+
("000000" & punto23(31 downto 6));
m2N30sumando30 <= "000000" & punto30(31 downto 6);
m2N30sumando31 <=( "00000" & punto31(31 downto 5))+
("000000" & punto31(31 downto 6));
m2N30sumando32 <=( "00000" & punto32(31 downto 5))+
("000000" & punto32(31 downto 6));
m2N30sumando33 <= "000000" & punto33(31 downto 6);
malla2N30 <= m2N30sumando00 + m2N30sumando01 +
m2N30sumando02 + m2N30sumando03 + m2N30sumando10 +
 m2N30sumando11 + m2N30sumando12 + m2N30sumando13 +
m2N30sumando20 + m2N30sumando21 + m2N30sumando22 +
m2N30sumando23 + m2N30sumando30 + m2N30sumando31 +
m2N30sumando32 + m2N30sumando33;

m2N31sumando00 <= "00000" & punto10(31 downto 5);
m2N31sumando01 <=( "0000" & punto11(31 downto 4))+
("00000" & punto11(31 downto 5));
m2N31sumando02 <=( "0000" & punto12(31 downto 4))+
 ("00000" & punto12(31 downto 5));
m2N31sumando03 <= "00000" & punto13(31 downto 5);
m2N31sumando10 <= "0000" & punto20(31 downto 4);
```

```
m2N31sumando11 <=( "000" & punto21(31 downto 3))+
("0000" & punto21(31 downto 4));
m2N31sumando12 <=( "000" & punto22(31 downto 3))+
("0000" & punto22(31 downto 4));
m2N31sumando13 <= "0000" & punto23(31 downto 4);
m2N31sumando20 <= "00000" & punto30(31 downto 5);
m2N31sumando21 <=( "0000" & punto31(31 downto 4))+
 ("00000" & punto31(31 downto 5));
m2N31sumando22 <=( "0000" & punto32(31 downto 4))+
("00000" & punto32(31 downto 5));
m2N31sumando23 <= "00000" & punto33(31 downto 5);
malla2N31 <= m2N31sumando00 + m2N31sumando01 +
m2N31sumando02 + m2N31sumando03 + m2N31sumando10 +
m2N31sumando11 + m2N31sumando12 + m2N31sumando13 +
m2N31sumando20 + m2N31sumando21 + m2N31sumando22 +
m2N31sumando23;

m2N32sumando00 <= "0000" & punto20(31 downto 4);
m2N32sumando01 <=( "000" & punto21(31 downto 3))+
("0000" & punto21(31 downto 4));
m2N32sumando02 <=( "000" & punto22(31 downto 3))+
("0000" & punto22(31 downto 4));
m2N32sumando03 <= "0000" & punto23(31 downto 4);
m2N32sumando10 <= "0000" & punto30(31 downto 4);
m2N32sumando11 <=( "000" & punto31(31 downto 3))+
("0000" & punto31(31 downto 4));
m2N32sumando12 <=( "000" & punto32(31 downto 3))+
("0000" & punto32(31 downto 4));
m2N32sumando13 <= "0000" & punto33(31 downto 4);
malla2N32 <= m2N32sumando00 + m2N32sumando01 +
m2N32sumando02 + m2N32sumando03 + m2N32sumando10 +
m2N32sumando11 + m2N32sumando12 + m2N32sumando13;

m2N33sumando00 <= "000" & punto30(31 downto 3);
m2N33sumando01 <=( "00" & punto31(31 downto 2))+
("000" & punto31(31 downto 3));
m2N33sumando02 <=( "00" & punto32(31 downto 2))+
("000" & punto32(31 downto 3));
m2N33sumando03 <= "000" & punto33(31 downto 3);
malla2N33 <= m2N33sumando00 + m2N33sumando01 +
m2N33sumando02 + m2N33sumando03;
```

Malla 3

```
m3N00sumando00 <= "000000" & punto00(31 downto 6);
m3N00sumando01 <=( "00000" & punto01(31 downto 5))+
("000000" & punto01(31 downto 6));
m3N00sumando02 <=( "00000" & punto02(31 downto 5))+
("000000" & punto02(31 downto 6));
m3N00sumando03 <= "000000" & punto03(31 downto 6);
m3N00sumando10 <=( "00000" & punto10(31 downto 5))+
("000000" & punto10(31 downto 6));
m3N00sumando11 <=( "000" & punto11(31 downto 3))+
("000000" & punto11(31 downto 6));
m3N00sumando12 <=( "000" & punto12(31 downto 3))+
("000000" & punto12(31 downto 6));
m3N00sumando13 <=( "00000" & punto13(31 downto 5))+
("000000" & punto13(31 downto 6));
m3N00sumando20 <=( "00000" & punto20(31 downto 5))+
("000000" & punto20(31 downto 6));
m3N00sumando21 <=( "000" & punto21(31 downto 3))+
("000000" & punto21(31 downto 6));
m3N00sumando22 <=( "000" & punto22(31 downto 3))+
("000000" & punto22(31 downto 6));
m3N00sumando23 <=( "00000" & punto23(31 downto 5))+
("000000" & punto23(31 downto 6));
m3N00sumando30 <= "000000" & punto30(31 downto 6);
m3N00sumando31 <=( "00000" & punto31(31 downto 5))+
("000000" & punto31(31 downto 6));
m3N00sumando32 <=( "00000" & punto32(31 downto 5))+
("000000" & punto32(31 downto 6));
m3N00sumando33 <= "000000" & punto33(31 downto 6);
malla3N00 <= m3N00sumando00 + m3N00sumando01 +
m3N00sumando02 + m3N00sumando03 + m3N00sumando10 +
m3N00sumando11 + m3N00sumando12 + m3N00sumando13 +
m3N00sumando20 + m3N00sumando21 + m3N00sumando22 +
m3N00sumando23 + m3N00sumando30 + m3N00sumando31 +
m3N00sumando32 + m3N00sumando33;

m3N01sumando00 <= "00000" & punto10(31 downto 5);
m3N01sumando01 <=( "0000" & punto11(31 downto 4))+
("00000" & punto11(31 downto 5));
m3N01sumando02 <=( "0000" & punto12(31 downto 4))+
```

```
("00000" & punto12(31 downto 5));
m3N01sumando03 <= "00000" & punto13(31 downto 5);
m3N01sumando10 <= "0000" & punto20(31 downto 4);
m3N01sumando11 <=( "000" & punto21(31 downto 3))+
("0000" & punto21(31 downto 4));
m3N01sumando12 <=( "000" & punto22(31 downto 3))+
("0000" & punto22(31 downto 4));
m3N01sumando13 <= "0000" & punto23(31 downto 4);
m3N01sumando20 <= "00000" & punto30(31 downto 5);
m3N01sumando21 <=( "0000" & punto31(31 downto 4))+
("00000" & punto31(31 downto 5));
m3N01sumando22 <=( "0000" & punto32(31 downto 4))+
("00000" & punto32(31 downto 5));
m3N01sumando23 <= "00000" & punto33(31 downto 5);
malla3N01 <= m3N01sumando00 + m3N01sumando01 +
m3N01sumando02 + m3N01sumando03 + m3N01sumando10 +
m3N01sumando11 + m3N01sumando12 + m3N01sumando13 +
m3N01sumando20 + m3N01sumando21 + m3N01sumando22 +
m3N01sumando23;

m3N02sumando00 <= "0000" & punto20(31 downto 4);
m3N02sumando01 <=( "000" & punto21(31 downto 3))+
("0000" & punto21(31 downto 4));
m3N02sumando02 <=( "000" & punto22(31 downto 3))+
("0000" & punto22(31 downto 4));
m3N02sumando03 <= "0000" & punto23(31 downto 4);
m3N02sumando10 <= "0000" & punto30(31 downto 4);
m3N02sumando11 <=( "000" & punto31(31 downto 3))+
("0000" & punto31(31 downto 4));
m3N02sumando12 <=( "000" & punto32(31 downto 3))+
("0000" & punto32(31 downto 4));
m3N02sumando13 <= "0000" & punto33(31 downto 4);
malla3N02 <= m3N02sumando00 + m3N02sumando01 +
m3N02sumando02 + m3N02sumando03 + m3N02sumando10 +
m3N02sumando11 + m3N02sumando12 + m3N02sumando13;

m3N03sumando00 <= "000" & punto30(31 downto 3);
m3N03sumando01 <=( "00" & punto31(31 downto 2))+
("000" & punto31(31 downto 3));
m3N03sumando02 <=( "00" & punto32(31 downto 2))+
("000" & punto32(31 downto 3));
```

```
m3N03sumando03 <= "000" & punto33(31 downto 3);
malla3N03 <= m3N03sumando00 + m3N03sumando01 +
m3N03sumando02 + m3N03sumando03;

m3N10sumando00 <= "00000" & punto01(31 downto 5);
m3N10sumando01 <= "0000" & punto02(31 downto 4);
m3N10sumando02 <= "00000" & punto03(31 downto 5);
m3N10sumando10 <=( "0000" & punto11(31 downto 4))+
("00000" & punto11(31 downto 5));
m3N10sumando11 <=( "000" & punto12(31 downto 3))+
("0000" & punto12(31 downto 4));
m3N10sumando12 <=( "0000" & punto13(31 downto 4))+
("00000" & punto13(31 downto 5));
m3N10sumando20 <=( "0000" & punto21(31 downto 4))+
("00000" & punto21(31 downto 5));
m3N10sumando21 <=( "000" & punto22(31 downto 3))+
("0000" & punto22(31 downto 4));
m3N10sumando22 <=( "0000" & punto23(31 downto 4))+
("00000" & punto23(31 downto 5));
m3N10sumando30 <= "00000" & punto31(31 downto 5);
m3N10sumando31 <= "0000" & punto32(31 downto 4);
m3N10sumando32 <= "00000" & punto33(31 downto 5);
malla3N10 <= m3N10sumando00 + m3N10sumando01 +
m3N10sumando02 + m3N10sumando10 + m3N10sumando11 +
m3N10sumando12 + m3N10sumando20 + m3N10sumando21 +
m3N10sumando22 + m3N10sumando30 + m3N10sumando31 +
m3N10sumando32;

m3N11sumando00 <= "0000" & punto11(31 downto 4);
m3N11sumando01 <= "000" & punto12(31 downto 3);
m3N11sumando02 <= "0000" & punto13(31 downto 4);
m3N11sumando10 <= "000" & punto21(31 downto 3);
m3N11sumando11 <= "00" & punto22(31 downto 2);
m3N11sumando12 <= "000" & punto23(31 downto 3);
m3N11sumando20 <= "0000" & punto31(31 downto 4);
m3N11sumando21 <= "000" & punto32(31 downto 3);
m3N11sumando22 <= "0000" & punto33(31 downto 4);
malla3N11 <= m3N11sumando00 + m3N11sumando01 +
m3N11sumando02 + m3N11sumando10 + m3N11sumando11 +
m3N11sumando12 + m3N11sumando20 + m3N11sumando21 +
m3N11sumando22;
```

```
m3N12sumando00 <= "000" & punto21(31 downto 3);
m3N12sumando01 <= "00" & punto22(31 downto 2);
m3N12sumando02 <= "000" & punto23(31 downto 3);
m3N12sumando10 <= "000" & punto31(31 downto 3);
m3N12sumando11 <= "00" & punto32(31 downto 2);
m3N12sumando12 <= "000" & punto33(31 downto 3);
malla3N12 <= m3N12sumando00 + m3N12sumando01 +
m3N12sumando02 + m3N12sumando10 + m3N12sumando11 +
m3N12sumando12;

m3N13sumando00 <= "00" & punto31(31 downto 2);
m3N13sumando01 <= "0" & punto32(31 downto 1);
m3N13sumando02 <= "00" & punto33(31 downto 2);
malla3N13 <= m3N13sumando00 + m3N13sumando01 +
m3N13sumando02;

m3N20sumando00 <= "0000" & punto02(31 downto 4);
m3N20sumando01 <= "0000" & punto03(31 downto 4);
m3N20sumando10 <=( "000" & punto12(31 downto 3))+
("0000" & punto12(31 downto 4));
m3N20sumando11 <=( "000" & punto13(31 downto 3))+
("0000" & punto13(31 downto 4));
m3N20sumando20 <=( "000" & punto22(31 downto 3))+
("0000" & punto22(31 downto 4));
m3N20sumando21 <=( "000" & punto23(31 downto 3))+
("0000" & punto23(31 downto 4));
m3N20sumando30 <= "0000" & punto32(31 downto 4);
m3N20sumando31 <= "0000" & punto33(31 downto 4);
malla3N20 <= m3N20sumando00 + m3N20sumando01 +
m3N20sumando10 + m3N20sumando11 + m3N20sumando20 +
 m3N20sumando21 + m3N20sumando30 + m3N20sumando31;

m3N21sumando00 <= "000" & punto12(31 downto 3);
m3N21sumando01 <= "000" & punto13(31 downto 3);
m3N21sumando10 <= "00" & punto22(31 downto 2);
m3N21sumando11 <= "00" & punto23(31 downto 2);
m3N21sumando20 <= "000" & punto32(31 downto 3);
m3N21sumando21 <= "000" & punto33(31 downto 3);
malla3N21 <= m3N21sumando00 + m3N21sumando01 +
m3N21sumando10 + m3N21sumando11 + m3N21sumando20 +
```

```
 m3N21sumando21;

m3N22sumando00 <= "00" & punto22(31 downto 2);
m3N22sumando01 <= "00" & punto23(31 downto 2);
m3N22sumando10 <= "00" & punto32(31 downto 2);
m3N22sumando11 <= "00" & punto33(31 downto 2);
malla3N22 <= m3N22sumando00 + m3N22sumando01 +
m3N22sumando10 + m3N22sumando11;

m3N23sumando00 <= "0" & punto32(31 downto 1);
m3N23sumando01 <= "0" & punto33(31 downto 1);
malla3N23 <= m3N23sumando00 + m3N23sumando01;

m3N30sumando00 <= "000" & punto03(31 downto 3);
m3N30sumando10 <=( "00" & punto13(31 downto 2))+
("000" & punto13(31 downto 3));
m3N30sumando20 <=( "00" & punto23(31 downto 2))+
("000" & punto23(31 downto 3));
m3N30sumando30 <= "000" & punto33(31 downto 3);
malla3N30 <= m3N30sumando00 + m3N30sumando10 +
m3N30sumando20 + m3N30sumando30;

m3N31sumando00 <= "00" & punto13(31 downto 2);
m3N31sumando10 <= "0" & punto23(31 downto 1);
m3N31sumando20 <= "00" & punto33(31 downto 2);
malla3N31 <= m3N31sumando00 + m3N31sumando10 +
m3N31sumando20;

m3N32sumando00 <= "0" & punto23(31 downto 1);
m3N32sumando10 <= "0" & punto33(31 downto 1);
malla3N32 <= m3N32sumando00 + m3N32sumando10;

m3N33sumando00 <=  punto33;
malla3N33 <= m3N33sumando00;

-------------------------
------- CONEXIONES -------
-------------------------

regS00 <= malla0N00;
regS01 <= malla0N01;
```

```
regS02 <= malla0N02;
regS03 <= malla0N03;
regS04 <= malla0N10;
regS05 <= malla0N11;
regS06 <= malla0N12;
regS07 <= malla0N13;
regS08 <= malla0N20;
regS09 <= malla0N21;
regS010 <= malla0N22;
regS011 <= malla0N23;
regS012 <= malla0N30;
regS013 <= malla0N31;
regS014 <= malla0N32;
regS015 <= malla0N33;
regS10 <= malla1N00;
regS11 <= malla1N01;
regS12 <= malla1N02;
regS13 <= malla1N03;
regS14 <= malla1N10;
regS15 <= malla1N11;
regS16 <= malla1N12;
regS17 <= malla1N13;
regS18 <= malla1N20;
regS19 <= malla1N21;
regS110 <= malla1N22;
regS111 <= malla1N23;
regS112 <= malla1N30;
regS113 <= malla1N31;
regS114 <= malla1N32;
regS115 <= malla1N33;
regS20 <= malla2N00;
regS21 <= malla2N01;
regS22 <= malla2N02;
regS23 <= malla2N03;
regS24 <= malla2N10;
regS25 <= malla2N11;
regS26 <= malla2N12;
regS27 <= malla2N13;
regS28 <= malla2N20;
regS29 <= malla2N21;
regS210 <= malla2N22;
```

```
regS211 <= malla2N23;
regS212 <= malla2N30;
regS213 <= malla2N31;
regS214 <= malla2N32;
regS215 <= malla2N33;
regS30 <= malla3N00;
regS31 <= malla3N01;
regS32 <= malla3N02;
regS33 <= malla3N03;
regS34 <= malla3N10;
regS35 <= malla3N11;
regS36 <= malla3N12;
regS37 <= malla3N13;
regS38 <= malla3N20;
regS39 <= malla3N21;
regS310 <= malla3N22;
regS311 <= malla3N23;
regS312 <= malla3N30;
regS313 <= malla3N31;
regS314 <= malla3N32;
regS315 <= malla3N33;

punto00 <= regE00;
punto01 <= regE01;
punto02 <= regE02;
punto03 <= regE03;
punto10 <= regE10;
punto11 <= regE11;
punto12 <= regE12;
punto13 <= regE13;
punto20 <= regE20;
punto21 <= regE21;
punto22 <= regE22;
punto23 <= regE23;
punto30 <= regE30;
punto31 <= regE31;
punto32 <= regE32;
punto33 <= regE33;

end imp;
```

D.2. *Hardware reducido*

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity divisor_mallas is
port (
clk : in std_logic;
rst : in std_logic;
entradaDatos : in std_logic_vector(31 downto 0);
salidaDatos : out std_logic_vector(31 downto 0);
entradaControl : in std_logic_vector(31 downto 0);
salidaControl : out std_logic_vector(31 downto 0)
);

end divisor_mallas;

architecture imp of divisor_mallas is

--Señales para los procesos del protocolo
signal datoOk, set, get, leido, escrito, rdy:std_logic;
signal datosIn, datosOut :
        std_logic_vector(31 downto 0);
type estados  is (espera,direccionando,
                  escribiendo,leyendo);
signal estado: estados;
signal contador: natural;
signal eDecodificador: std_logic;
signal eMultiplexor: std_logic;

--Señales para conectar el divisor con el protocolo
-- (datos de entrada)
signal regE00, regE01, regE02, regE03, regE10, regE11,
        regE12, regE13, regE20, regE21, regE22, regE23,
        regE30, regE31, regE32, regE33 :
         std_logic_vector(31 downto 0);
```

```
--Señales para conectar el divisor con el protocolo
-- (datos de salida)
signal regS00, regS01, regS02, regS03, regS04,
       regS05, regS06, regS07, regS08,
       regS09, regS010, regS011, regS012,
       regS013, regS014, regS015, regS10,
       regS11, regS12, regS13, regS14, regS15,
       regS16, regS17, regS18, regS19,
       regS110, regS111, regS112, regS113, regS114,
       regS115, regS20, regS21,
       regS22, regS23, regS24, regS25, regS26, regS27,
       regS28, regS29, regS210,
       regS211, regS212, regS213, regS214, regS215,
       regS30, regS31, regS32,
       regS33, regS34, regS35, regS36, regS37, regS38,
       regS39, regS310, regS311,
       regS312, regS313, regS314, regS315 :
       std_logic_vector(31 downto 0);

--Señales para los operandos resultado del divisor.
signal malla0N00, malla0N01, malla0N02, malla0N03,
       malla0N10, malla0N11, malla0N12, malla0N13,
       malla0N20, malla0N21, malla0N22, malla0N23,
       malla0N30, malla0N31, malla0N32,
       malla0N33, malla1N00, malla1N01, malla1N02,
       malla1N03, malla1N10, malla1N11,
       malla1N12, malla1N13, malla1N20, malla1N21,
       malla1N22, malla1N23, malla1N30,
       malla1N31, malla1N32, malla1N33, malla2N00,
       malla2N01, malla2N02, malla2N03,
       malla2N10, malla2N11, malla2N12, malla2N13,
       malla2N20, malla2N21, malla2N22,
       malla2N23, malla2N30, malla2N31, malla2N32,
       malla2N33, malla3N00, malla3N01,
       malla3N02, malla3N03, malla3N10, malla3N11,
       malla3N12, malla3N13, malla3N20,
       malla3N21, malla3N22, malla3N23, malla3N30,
       malla3N31, malla3N32, malla3N33 :
       std_logic_vector(31 downto 0);
```

```
--Señales intermedias
signal puntoDes000, suma_000, puntoDes200, puntoDes210,
       suma_210_200, puntoDes300, puntoDes320,
       suma_210_300_320, puntoDes400, puntoDes310,
       puntoDes410, puntoDes420, puntoDes430,
       suma_420_430_400_410_310_320, puntoDes201,
       suma_200_201, puntoDes301, puntoDes311,
       suma_311_301_300_310, puntoDes401, puntoDes421,
       suma_401_421_311_420_400_310, puntoDes500,
       puntoDes501, puntoDes510, puntoDes411,
       puntoDes511, puntoDes520, puntoDes521,
       puntoDes530, puntoDes531,suma_411_501_511_521_
                    531_421_420_500_510_520_530_410,
       puntoDes302, suma_302_300_201, puntoDes402,
       puntoDes412, suma_311_301_400_410_412_402,
       puntoDes502, puntoDes522,
       suma_401_421_311_500_520_410_412_502_522,
       puntoDes600, puntoDes602, puntoDes610,
       puntoDes512, puntoDes612, puntoDes620,
       puntoDes622, puntoDes630, puntoDes632,
       suma_411_501_511_521_531_421_510_520_622_632_
                    512_522_602_612_610_600_630_620,
       puntoDes403,
       suma_401_302_403_301_400_402, puntoDes503,
       puntoDes513, suma_401_411_501_511_513_503_500_
                         510_412_502_512_402,
       puntoDes601, puntoDes603, puntoDes621,
       puntoDes623, suma_601_411_501_511_521_621_510_
          622_502_512_522_602_603_513_600_623_620_412,
       puntoDes700, puntoDes701, puntoDes702,
       puntoDes703, puntoDes710, puntoDes711,
       puntoDes712, puntoDes613, puntoDes713,
       puntoDes720, puntoDes721, puntoDes422,
       puntoDes722, puntoDes723, puntoDes730,
       puntoDes631, puntoDes731, puntoDes732,
       puntoDes733, suma_601_411_421_631_632_602_720_
       730_700_710_613_623_731_721_711_701_703_713_
              723_733_610_702_712_722_732_620_412_422,
       puntoDes202, puntoDes303, suma_202_301_303,
       puntoDes312, puntoDes413, suma_401_411_413_302_
       312_403, puntoDes523, suma_523_411_501_521_413_
```

```
        312_503_402_422 , puntoDes611 , puntoDes532 ,
        puntoDes633 , suma_601_611_511_521_621_631_502_
        512_522_532_523_613_603_513_633_623_412_422 ,
        puntoDes203 , suma_202_203 , puntoDes313 ,
        suma_302_312_303_313 , puntoDes423 ,
        suma_312_403_423_313_402_422 , puntoDes533 ,
        suma_533_523_513_413_503_423_412_502_512_522_
                                                532_422 ,
        puntoDes003 , suma_003 , puntoDes213 ,
        suma_213_203 , puntoDes323 , suma_213_323_303 ,
        puntoDes433 , suma_413_403_423_433_323_313 ,
        puntoDes220 , puntoDes330 , suma_220_310_330 ,
        puntoDes230 , suma_230_220 , puntoDes030 ,
        suma_030 , puntoDes321 , puntoDes431 ,
        suma_411_431_430_321_410_320 , puntoDes331 ,
        suma_331_321_320_330 , puntoDes231 ,
        suma_230_231 , suma_411_431_420_321_510_530_512_
        532_422 , puntoDes432 ,
        suma_331_420_430_321_432_422 , puntoDes332 ,
        suma_332_231_330 , suma_611_511_521_531_421_631_
        520_632_512_522_532_612_523_613_610_633_
        630_422 , suma_533_523_521_531_421_431_520_530_
        432_522_532_422 , suma_332_431_331_433_430_432 ,
        puntoDes322 , suma_533_511_322_531_421_513_423_
        412_432 , suma_322_332_421_431_423_433 ,
        puntoDes232 , puntoDes333 , suma_232_331_333 ,
        suma_322_413_433_323_412_432 ,
        suma_322_332_323_333 , puntoDes233 ,
        suma_232_233 , puntoDes223 , suma_223_333_313 ,
        suma_233_223 , puntoDes033 , suma_033 :
         std_logic_vector(31 downto 0);

begin

-------------------------------
 ------- PROTOCOLO -------
-------------------------------
-- Entradas de control:
set <= entradaControl(17);
get <= entradaControl(18);
leido <= entradaControl(19);
```

```
escrito <= entradaControl(20);

-- Salidas de control:
salidaControl(0) <= datoOk;
salidaControl(1) <= rdy;
salidaControl(31 downto 2) <= (others => '0');

-- Datos
salidaDatos <= datosOut;
datosIn <= entradaDatos;
controlador:
process(clk, rst)
begin
        rdy <= '0';
        datoOk <= '0';
        eDecodificador <= '0';
        eMultiplexor <= '0';
        if (rst = '0') then
                estado <= espera;
                contador <= 0;
        elsif (clk'event and clk = '1') then
                case estado is
                  when espera =>
                     contador <= 0;
                     if(set = '1') then
                        estado <= escribiendo;
                     elsif(get = '1') then
                        estado <= leyendo;
                     end if;

                   when escribiendo =>
                      if(escrito = '1')then
                         if(contador = 15) then
                            estado <= espera;
                         else
                            estado <= direccionando;
                            contador <= contador + 1;
                          end if;
                      end if;

                   when direccionando =>
```

```
                        if(set='1') then
                          estado<=escribiendo;
                        elsif(get='1') then
                          estado<=leyendo;
                        end if;

                      when leyendo =>
                          if(leido = '1') then
                             if(contador = 63) then
                                estado <= espera;
                             else
                                estado <= direccionando;
                                contador <= contador + 1;
                             end if;
                          end if;

                        when others =>
                          null;
                end case;
        end if;

        case estado is
                when espera =>
                        rdy <= '1';
                when escribiendo =>
                        eDecodificador <= '1';
                        datoOk <= '1';
                when leyendo =>
                        eMultiplexor <= '1';
                        datoOk <= '1';
                when direccionando =>
                        null;
                when others =>
                        null;
        end case;
end process controlador;

Decodificador:
process (clk,rst)
begin
        if(rst='0') then
```

```
                regE00 <= (others => '0');
                regE01 <= (others => '0');
                regE02 <= (others => '0');
                regE03 <= (others => '0');
                regE10 <= (others => '0');
                regE11 <= (others => '0');
                regE12 <= (others => '0');
                regE13 <= (others => '0');
                regE20 <= (others => '0');
                regE21 <= (others => '0');
                regE22 <= (others => '0');
                regE23 <= (others => '0');
                regE30 <= (others => '0');
                regE31 <= (others => '0');
                regE32 <= (others => '0');
                regE33 <= (others => '0');
        elsif(clk'event and clk='1') then
                if(eDecodificador = '1') then
                    case contador is
                        when 0 => regE00 <= datosIn;
                        when 1 => regE01 <= datosIn;
                        when 2 => regE02 <= datosIn;
                        when 3 => regE03 <= datosIn;
                        when 4 => regE10 <= datosIn;
                        when 5 => regE11 <= datosIn;
                        when 6 => regE12 <= datosIn;
                        when 7 => regE13 <= datosIn;
                        when 8 => regE20 <= datosIn;
                        when 9 => regE21 <= datosIn;
                        when 10 => regE22 <= datosIn;
                        when 11 => regE23 <= datosIn;
                        when 12 => regE30 <= datosIn;
                        when 13 => regE31 <= datosIn;
                        when 14 => regE32 <= datosIn;
                        when 15 => regE33 <= datosIn;
                        when others =>null;
                    end case;
                end if;
        end if;
end process;
```

```
multiplexor:
process(rst,clk)
begin
        datosOut <= (others => '0');
        if(eMultiplexor='1') then
                case contador is
                        when 0 => datosOut <= regS00;
                        when 1 => datosOut <= regS01;
                        when 2 => datosOut <= regS02;
                        when 3 => datosOut <= regS03;
                        when 4 => datosOut <= regS04;
                        when 5 => datosOut <= regS05;
                        when 6 => datosOut <= regS06;
                        when 7 => datosOut <= regS07;
                        when 8 => datosOut <= regS08;
                        when 9 => datosOut <= regS09;
                        when 10 => datosOut <= regS010;
                        when 11 => datosOut <= regS011;
                        when 12 => datosOut <= regS012;
                        when 13 => datosOut <= regS013;
                        when 14 => datosOut <= regS014;
                        when 15 => datosOut <= regS015;
                        when 16 => datosOut <= regS10;
                        when 17 => datosOut <= regS11;
                        when 18 => datosOut <= regS12;
                        when 19 => datosOut <= regS13;
                        when 20 => datosOut <= regS14;
                        when 21 => datosOut <= regS15;
                        when 22 => datosOut <= regS16;
                        when 23 => datosOut <= regS17;
                        when 24 => datosOut <= regS18;
                        when 25 => datosOut <= regS19;
                        when 26 => datosOut <= regS110;
                        when 27 => datosOut <= regS111;
                        when 28 => datosOut <= regS112;
                        when 29 => datosOut <= regS113;
                        when 30 => datosOut <= regS114;
                        when 31 => datosOut <= regS115;
                        when 32 => datosOut <= regS20;
                        when 33 => datosOut <= regS21;
                        when 34 => datosOut <= regS22;
```

```
                    when 35 => datosOut <= regS23;
                    when 36 => datosOut <= regS24;
                    when 37 => datosOut <= regS25;
                    when 38 => datosOut <= regS26;
                    when 39 => datosOut <= regS27;
                    when 40 => datosOut <= regS28;
                    when 41 => datosOut <= regS29;
                    when 42 => datosOut <= regS210;
                    when 43 => datosOut <= regS211;
                    when 44 => datosOut <= regS212;
                    when 45 => datosOut <= regS213;
                    when 46 => datosOut <= regS214;
                    when 47 => datosOut <= regS215;
                    when 48 => datosOut <= regS30;
                    when 49 => datosOut <= regS31;
                    when 50 => datosOut <= regS32;
                    when 51 => datosOut <= regS33;
                    when 52 => datosOut <= regS34;
                    when 53 => datosOut <= regS35;
                    when 54 => datosOut <= regS36;
                    when 55 => datosOut <= regS37;
                    when 56 => datosOut <= regS38;
                    when 57 => datosOut <= regS39;
                    when 58 => datosOut <= regS310;
                    when 59 => datosOut <= regS311;
                    when 60 => datosOut <= regS312;
                    when 61 => datosOut <= regS313;
                    when 62 => datosOut <= regS314;
                    when 63 => datosOut <= regS315;
                    when others =>
                        datosOut <= (others => '0');
                end case;
        end if;
end process;

-------------------------
 ------- DIVISOR -------
-------------------------

--Desplazamientos
```

```
puntoDes000 <= regE00;
puntoDes200 <= "00" & regE00(31 downto 2);
puntoDes210 <= "00" & regE10(31 downto 2);
puntoDes300 <= "000" & regE00(31 downto 3);
puntoDes320 <= "000" & regE20(31 downto 3);
puntoDes400 <= "0000" & regE00(31 downto 4);
puntoDes310 <= "000" & regE10(31 downto 3);
puntoDes410 <= "0000" & regE10(31 downto 4);
puntoDes420 <= "0000" & regE20(31 downto 4);
puntoDes430 <= "0000" & regE30(31 downto 4);
puntoDes201 <= "00" & regE01(31 downto 2);
puntoDes301 <= "000" & regE01(31 downto 3);
puntoDes311 <= "000" & regE11(31 downto 3);
puntoDes401 <= "0000" & regE01(31 downto 4);
puntoDes421 <= "0000" & regE21(31 downto 4);
puntoDes500 <= "00000" & regE00(31 downto 5);
puntoDes501 <= "00000" & regE01(31 downto 5);
puntoDes510 <= "00000" & regE10(31 downto 5);
puntoDes411 <= "0000" & regE11(31 downto 4);
puntoDes511 <= "00000" & regE11(31 downto 5);
puntoDes520 <= "00000" & regE20(31 downto 5);
puntoDes521 <= "00000" & regE21(31 downto 5);
puntoDes530 <= "00000" & regE30(31 downto 5);
puntoDes531 <= "00000" & regE31(31 downto 5);
puntoDes302 <= "000" & regE02(31 downto 3);
puntoDes402 <= "0000" & regE02(31 downto 4);
puntoDes412 <= "0000" & regE12(31 downto 4);
puntoDes502 <= "00000" & regE02(31 downto 5);
puntoDes522 <= "00000" & regE22(31 downto 5);
puntoDes600 <= "000000" & regE00(31 downto 6);
puntoDes602 <= "000000" & regE02(31 downto 6);
puntoDes610 <= "000000" & regE10(31 downto 6);
puntoDes512 <= "00000" & regE12(31 downto 5);
puntoDes612 <= "000000" & regE12(31 downto 6);
puntoDes620 <= "000000" & regE20(31 downto 6);
puntoDes622 <= "000000" & regE22(31 downto 6);
puntoDes630 <= "000000" & regE30(31 downto 6);
puntoDes632 <= "000000" & regE32(31 downto 6);
puntoDes403 <= "0000" & regE03(31 downto 4);
puntoDes503 <= "00000" & regE03(31 downto 5);
puntoDes513 <= "00000" & regE13(31 downto 5);
```

```
puntoDes601 <= "000000" & regE01(31 downto 6);
puntoDes603 <= "000000" & regE03(31 downto 6);
puntoDes621 <= "000000" & regE21(31 downto 6);
puntoDes623 <= "000000" & regE23(31 downto 6);
puntoDes700 <= "0000000" & regE00(31 downto 7);
puntoDes701 <= "0000000" & regE01(31 downto 7);
puntoDes702 <= "0000000" & regE02(31 downto 7);
puntoDes703 <= "0000000" & regE03(31 downto 7);
puntoDes710 <= "0000000" & regE10(31 downto 7);
puntoDes711 <= "0000000" & regE11(31 downto 7);
puntoDes712 <= "0000000" & regE12(31 downto 7);
puntoDes613 <= "000000" & regE13(31 downto 6);
puntoDes713 <= "0000000" & regE13(31 downto 7);
puntoDes720 <= "0000000" & regE20(31 downto 7);
puntoDes721 <= "0000000" & regE21(31 downto 7);
puntoDes422 <= "0000" & regE22(31 downto 4);
puntoDes722 <= "0000000" & regE22(31 downto 7);
puntoDes723 <= "0000000" & regE23(31 downto 7);
puntoDes730 <= "0000000" & regE30(31 downto 7);
puntoDes631 <= "000000" & regE31(31 downto 6);
puntoDes731 <= "0000000" & regE31(31 downto 7);
puntoDes732 <= "0000000" & regE32(31 downto 7);
puntoDes733 <= "0000000" & regE33(31 downto 7);
puntoDes202 <= "00" & regE02(31 downto 2);
puntoDes303 <= "000" & regE03(31 downto 3);
puntoDes312 <= "000" & regE12(31 downto 3);
puntoDes413 <= "0000" & regE13(31 downto 4);
puntoDes523 <= "00000" & regE23(31 downto 5);
puntoDes611 <= "000000" & regE11(31 downto 6);
puntoDes532 <= "00000" & regE32(31 downto 5);
puntoDes633 <= "000000" & regE33(31 downto 6);
puntoDes203 <= "00" & regE03(31 downto 2);
puntoDes313 <= "000" & regE13(31 downto 3);
puntoDes423 <= "0000" & regE23(31 downto 4);
puntoDes533 <= "00000" & regE33(31 downto 5);
puntoDes003 <= regE03;
puntoDes213 <= "00" & regE13(31 downto 2);
puntoDes323 <= "000" & regE23(31 downto 3);
puntoDes433 <= "0000" & regE33(31 downto 4);
puntoDes220 <= "00" & regE20(31 downto 2);
puntoDes330 <= "000" & regE30(31 downto 3);
```

```
puntoDes230 <= "00" & regE30(31 downto 2);
puntoDes030 <= regE30;
puntoDes321 <= "000" & regE21(31 downto 3);
puntoDes431 <= "0000" & regE31(31 downto 4);
puntoDes331 <= "000" & regE31(31 downto 3);
puntoDes231 <= "00" & regE31(31 downto 2);
puntoDes432 <= "0000" & regE32(31 downto 4);
puntoDes332 <= "000" & regE32(31 downto 3);
puntoDes322 <= "000" & regE22(31 downto 3);
puntoDes232 <= "00" & regE32(31 downto 2);
puntoDes333 <= "000" & regE33(31 downto 3);
puntoDes233 <= "00" & regE33(31 downto 2);
puntoDes223 <= "00" & regE23(31 downto 2);
puntoDes033 <= regE33;
```

—*Sumas*

```
suma_000 <= puntoDes000;
suma_210_200 <= puntoDes210 + puntoDes200;
suma_210_300_320 <= puntoDes210 + puntoDes300 +
puntoDes320;
suma_420_430_400_410_310_320 <= puntoDes420 +
puntoDes430 + puntoDes400 + puntoDes410 + puntoDes310 +
puntoDes320;
suma_200_201 <= puntoDes200 + puntoDes201;
suma_311_301_300_310 <= puntoDes311 + puntoDes301 +
puntoDes300 + puntoDes310;
suma_401_421_311_420_400_310 <= puntoDes401 +
puntoDes421 + puntoDes311 + puntoDes420 + puntoDes400 +
puntoDes310;
suma_411_501_511_521_531_421_420_500_510_520_530_410 <=
puntoDes411 + puntoDes501 + puntoDes511 + puntoDes521 +
puntoDes531 + puntoDes421 + puntoDes420 + puntoDes500 +
puntoDes510 + puntoDes520 + puntoDes530 + puntoDes410;
suma_302_300_201 <= puntoDes302 + puntoDes300 +
puntoDes201;
suma_311_301_400_410_412_402 <= puntoDes311 +
puntoDes301 + puntoDes400 + puntoDes410 + puntoDes412 +
puntoDes402;
suma_401_421_311_500_520_410_412_502_522 <= puntoDes401
 + puntoDes421 + puntoDes311 + puntoDes500 +
```

```
puntoDes520 + puntoDes410 + puntoDes412 + puntoDes502 +
puntoDes522;
suma_411_501_511_521_531_421_510_520_622_632_512_522_
602_612_610_600_630_620 <= puntoDes411 + puntoDes501 +
puntoDes511 + puntoDes521 + puntoDes531 + puntoDes421 +
puntoDes510 + puntoDes520 + puntoDes622 + puntoDes632 +
puntoDes512 + puntoDes522 + puntoDes602 + puntoDes612 +
puntoDes610 + puntoDes600 + puntoDes630 + puntoDes620;
suma_401_302_403_301_400_402 <= puntoDes401 +
puntoDes302 + puntoDes403 + puntoDes301 + puntoDes400 +
puntoDes402;
suma_401_411_501_511_513_503_500_510_412_502_512_402 <=
puntoDes401 + puntoDes411 + puntoDes501 + puntoDes511 +
puntoDes513 + puntoDes503 + puntoDes500 + puntoDes510 +
puntoDes412 + puntoDes502 + puntoDes512 + puntoDes402;
suma_601_411_501_511_521_621_510_622_502_512_522_602_
603_513_600_623_620_412 <= puntoDes601 + puntoDes411 +
puntoDes501 + puntoDes511 + puntoDes521 + puntoDes621 +
puntoDes510 + puntoDes622 + puntoDes502 + puntoDes512 +
puntoDes522 + puntoDes602 + puntoDes603 + puntoDes513 +
puntoDes600 + puntoDes623 + puntoDes620 + puntoDes412;
suma_601_411_421_631_632_602_720_730_700_710_613_623_
731_721_711_701_703_713_723_733_610_702_712_722_732_
620_412_422 <= puntoDes601 + puntoDes411 + puntoDes421
 + puntoDes631 + puntoDes632 + puntoDes602 +
puntoDes720 + puntoDes730 + puntoDes700 + puntoDes710 +
 puntoDes613 + puntoDes623 + puntoDes731 + puntoDes721
 + puntoDes711 + puntoDes701 + puntoDes703 +
puntoDes713 + puntoDes723 + puntoDes733 + puntoDes610 +
puntoDes702 + puntoDes712 + puntoDes722 + puntoDes732 +
puntoDes620 + puntoDes412 + puntoDes422;
suma_202_301_303 <= puntoDes202 + puntoDes301 +
puntoDes303;
suma_401_411_413_302_312_403 <= puntoDes401 +
puntoDes411 + puntoDes413 + puntoDes302 + puntoDes312 +
puntoDes403;
suma_523_411_501_521_413_312_503_402_422 <= puntoDes523
 + puntoDes411 + puntoDes501 + puntoDes521 +
puntoDes413 + puntoDes312 + puntoDes503 + puntoDes402 +
puntoDes422;
suma_601_611_511_521_621_631_502_512_522_532_523_613_
```

```
603_513_633_623_412_422 <= puntoDes601 + puntoDes611 +
 puntoDes511 + puntoDes521 + puntoDes621 + puntoDes631
 + puntoDes502 + puntoDes512 + puntoDes522 +
puntoDes532 + puntoDes523 + puntoDes613 + puntoDes603 +
puntoDes513 + puntoDes633 + puntoDes623 + puntoDes412 +
puntoDes422;
suma_202_203 <= puntoDes202 + puntoDes203;
suma_302_312_303_313 <= puntoDes302 + puntoDes312 +
puntoDes303 + puntoDes313;
suma_312_403_423_313_402_422 <= puntoDes312 +
puntoDes403 + puntoDes423 + puntoDes313 + puntoDes402 +
puntoDes422;
suma_533_523_513_413_503_423_412_502_512_522_532_422 <=
puntoDes533 + puntoDes523 + puntoDes513 + puntoDes413 +
puntoDes503 + puntoDes423 + puntoDes412 + puntoDes502 +
puntoDes512 + puntoDes522 + puntoDes532 + puntoDes422;
suma_003 <= puntoDes003;
suma_213_203 <= puntoDes213 + puntoDes203;
suma_213_323_303 <= puntoDes213 + puntoDes323 +
puntoDes303;
suma_413_403_423_433_323_313 <= puntoDes413 +
puntoDes403 + puntoDes423 + puntoDes433 + puntoDes323 +
puntoDes313;
suma_220_310_330 <= puntoDes220 + puntoDes310 +
puntoDes330;
suma_230_220 <= puntoDes230 + puntoDes220;
suma_030 <= puntoDes030;
suma_411_431_430_321_410_320 <= puntoDes411 +
puntoDes431 + puntoDes430 + puntoDes321 +
puntoDes410 + puntoDes320;
suma_331_321_320_330 <= puntoDes331 + puntoDes321 +
puntoDes320 + puntoDes330;
suma_230_231 <= puntoDes230 + puntoDes231;
suma_411_431_420_321_510_530_512_532_422 <= puntoDes411
 + puntoDes431 + puntoDes420 + puntoDes321 +
puntoDes510 + puntoDes530 + puntoDes512 + puntoDes532 +
puntoDes422;
suma_331_420_430_321_432_422 <= puntoDes331 +
puntoDes420 + puntoDes430 + puntoDes321 + puntoDes432 +
puntoDes422;
suma_332_231_330 <= puntoDes332 + puntoDes231 +
```

```
puntoDes330;
suma_611_511_521_531_421_631_520_632_512_522_532_612_
523_613_610_633_630_422 <= puntoDes611 + puntoDes511 +
puntoDes521 + puntoDes531 + puntoDes421 + puntoDes631 +
puntoDes520 + puntoDes632 + puntoDes512 + puntoDes522 +
puntoDes532 + puntoDes612 + puntoDes523 + puntoDes613 +
puntoDes610 + puntoDes633 + puntoDes630 + puntoDes422;
suma_533_523_521_531_421_431_520_530_432_522_532_422 <=
puntoDes533 + puntoDes523 + puntoDes521 + puntoDes531 +
puntoDes421 + puntoDes431 + puntoDes520 + puntoDes530 +
puntoDes432 + puntoDes522 + puntoDes532 + puntoDes422;
suma_332_431_331_433_430_432 <= puntoDes332 +
puntoDes431 + puntoDes331 + puntoDes433 + puntoDes430 +
puntoDes432;
suma_533_511_322_531_421_513_423_412_432 <= puntoDes533
 + puntoDes511 + puntoDes322 + puntoDes531 +
puntoDes421 + puntoDes513 + puntoDes423 + puntoDes412 +
puntoDes432;
suma_322_332_421_431_423_433 <= puntoDes322 +
puntoDes332 + puntoDes421 + puntoDes431 + puntoDes423 +
puntoDes433;
suma_232_331_333 <= puntoDes232 + puntoDes331 +
puntoDes333;
suma_322_413_433_323_412_432 <= puntoDes322 +
puntoDes413 + puntoDes433 + puntoDes323 + puntoDes412 +
puntoDes432;
suma_322_332_323_333 <= puntoDes322 + puntoDes332 +
puntoDes323 + puntoDes333;
suma_232_233 <= puntoDes232 + puntoDes233;
suma_223_333_313 <= puntoDes223 + puntoDes333 +
 puntoDes313;
suma_233_223 <= puntoDes233 + puntoDes223;
suma_033 <= puntoDes033;

--Resultados

malla0N00 <= suma_000;
malla0N01 <= suma_210_200;
malla0N02 <= suma_210_300_320;
malla0N03 <= suma_420_430_400_410_310_320;
malla0N10 <= suma_200_201;
```

```
malla0N11 <= suma_311_301_300_310;
malla0N12 <= suma_401_421_311_420_400_310;
malla0N13 <= suma_411_501_511_521_531_421_420_500_510_
520_530_410;
malla0N20 <= suma_302_300_201;
malla0N21 <= suma_311_301_400_410_412_402;
malla0N22 <= suma_401_421_311_500_520_410_412_502_522;
malla0N23 <= suma_411_501_511_521_531_421_510_520_622_
632_512_522_602_612_610_600_630_620;
malla0N30 <= suma_401_302_403_301_400_402;
malla0N31 <= suma_401_411_501_511_513_503_500_510_412_
502_512_402;
malla0N32 <= suma_601_411_501_511_521_621_510_622_502_
512_522_602_603_513_600_623_620_412;
malla0N33 <= suma_601_411_421_631_632_602_720_730_700_
710_613_623_731_721_711_701_703_713_723_733_610_702_
712_722_732_620_412_422;
malla1N00 <= suma_401_302_403_301_400_402;
malla1N01 <= suma_401_411_501_511_513_503_500_510_412_
502_512_402;
malla1N02 <= suma_601_411_501_511_521_621_510_622_502_
512_522_602_603_513_600_623_620_412;
malla1N03 <= suma_601_411_421_631_632_602_720_730_700_
710_613_623_731_721_711_701_703_713_723_733_610_702_
712_722_732_620_412_422;
malla1N10 <= suma_202_301_303;
malla1N11 <= suma_401_411_413_302_312_403;
malla1N12 <= suma_523_411_501_521_413_312_503_402_422;
malla1N13 <= suma_601_611_511_521_621_631_502_512_522_
532_523_613_603_513_633_623_412_422;
malla1N20 <= suma_202_203;
malla1N21 <= suma_302_312_303_313;
malla1N22 <= suma_312_403_423_313_402_422;
malla1N23 <= suma_533_523_513_413_503_423_412_502_512_
522_532_422;
malla1N30 <= suma_003;
malla1N31 <= suma_213_203;
malla1N32 <= suma_213_323_303;
malla1N33 <= suma_413_403_423_433_323_313;
malla2N00 <= suma_420_430_400_410_310_320;
malla2N01 <= suma_220_310_330;
```

```
malla2N02 <= suma_230_220;
malla2N03 <= suma_030;
malla2N10 <= suma_411_501_511_521_531_421_420_500_510_
520_530_410;
malla2N11 <= suma_411_431_430_321_410_320;
malla2N12 <= suma_331_321_320_330;
malla2N13 <= suma_230_231;
malla2N20 <= suma_411_501_511_521_531_421_510_520_622_
632_512_522_602_612_610_600_630_620;
malla2N21 <= suma_411_431_420_321_510_530_512_532_422;
malla2N22 <= suma_331_420_430_321_432_422;
malla2N23 <= suma_332_231_330;
malla2N30 <= suma_601_411_421_631_632_602_720_730_700_
710_613_623_731_721_711_701_703_713_723_733_610_702_
712_722_732_620_412_422;
malla2N31 <= suma_611_511_521_531_421_631_520_632_
512_522_532_612_523_613_610_633_630_422;
malla2N32 <= suma_533_523_521_531_421_431_520_530_432_
522_532_422;
malla2N33 <= suma_332_431_331_433_430_432;
malla3N00 <= suma_601_411_421_631_632_602_720_730_700_
710_613_623_731_721_711_701_703_713_723_733_610_702_
712_722_732_620_412_422;
malla3N01 <= suma_611_511_521_531_421_631_520_632_512_
522_532_612_523_613_610_633_630_422;
malla3N02 <= suma_533_523_521_531_421_431_520_530_432_
522_532_422;
malla3N03 <= suma_332_431_331_433_430_432;
malla3N10 <= suma_601_611_511_521_621_631_502_512_522_
532_523_613_603_513_633_623_412_422;
malla3N11 <= suma_533_511_322_531_421_513_423_412_432;
malla3N12 <= suma_322_332_421_431_423_433;
malla3N13 <= suma_232_331_333;
malla3N20 <= suma_533_523_513_413_503_423_412_502_512_
522_532_422;
malla3N21 <= suma_322_413_433_323_412_432;
malla3N22 <= suma_322_332_323_333;
malla3N23 <= suma_232_233;
malla3N30 <= suma_413_403_423_433_323_313;
malla3N31 <= suma_223_333_313;
malla3N32 <= suma_233_223;
```

```
malla3N33 <= suma_033;

-------------------------------
------- CONEXIONES -------
-------------------------------

regS00 <= malla0N00;
regS01 <= malla0N01;
regS02 <= malla0N02;
regS03 <= malla0N03;
regS04 <= malla0N10;
regS05 <= malla0N11;
regS06 <= malla0N12;
regS07 <= malla0N13;
regS08 <= malla0N20;
regS09 <= malla0N21;
regS010 <= malla0N22;
regS011 <= malla0N23;
regS012 <= malla0N30;
regS013 <= malla0N31;
regS014 <= malla0N32;
regS015 <= malla0N33;
regS10 <= malla1N00;
regS11 <= malla1N01;
regS12 <= malla1N02;
regS13 <= malla1N03;
regS14 <= malla1N10;
regS15 <= malla1N11;
regS16 <= malla1N12;
regS17 <= malla1N13;
regS18 <= malla1N20;
regS19 <= malla1N21;
regS110 <= malla1N22;
regS111 <= malla1N23;
regS112 <= malla1N30;
regS113 <= malla1N31;
regS114 <= malla1N32;
regS115 <= malla1N33;
regS20 <= malla2N00;
regS21 <= malla2N01;
regS22 <= malla2N02;
```

```
regS23 <= malla2N03;
regS24 <= malla2N10;
regS25 <= malla2N11;
regS26 <= malla2N12;
regS27 <= malla2N13;
regS28 <= malla2N20;
regS29 <= malla2N21;
regS210 <= malla2N22;
regS211 <= malla2N23;
regS212 <= malla2N30;
regS213 <= malla2N31;
regS214 <= malla2N32;
regS215 <= malla2N33;
regS30 <= malla3N00;
regS31 <= malla3N01;
regS32 <= malla3N02;
regS33 <= malla3N03;
regS34 <= malla3N10;
regS35 <= malla3N11;
regS36 <= malla3N12;
regS37 <= malla3N13;
regS38 <= malla3N20;
regS39 <= malla3N21;
regS310 <= malla3N22;
regS311 <= malla3N23;
regS312 <= malla3N30;
regS313 <= malla3N31;
regS314 <= malla3N32;
regS315 <= malla3N33;

end imp;
```

D.3. *Hardware con árboles*

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
entity divisor_mallas is
port (
clk : in std_logic;
rst : in std_logic;
entradaDatos : in std_logic_vector(31 downto 0);
salidaDatos : out std_logic_vector(31 downto 0);
entradaControl : in std_logic_vector(31 downto 0);
salidaControl : out std_logic_vector(31 downto 0)
);

end divisor_mallas;

architecture imp of divisor_mallas is

--Señales para los procesos del protocolo
signal datoOk, set, get, leido, escrito, rdy:std_logic;
signal datosIn, datosOut :
                        std_logic_vector(31 downto 0);
type estados  is (espera,direccionando,
                  escribiendo,leyendo);
signal estado: estados;
signal contador: natural;
signal eDecodificador: std_logic;
signal eMultiplexor: std_logic;

--Señales para conectar el divisor con el protocolo
--(datos de entrada)
signal regE00, regE01, regE02, regE03, regE10, regE11,
       regE12, regE13, regE20, regE21, regE22, regE23,
       regE30, regE31, regE32, regE33 :
       std_logic_vector(31 downto 0);

--Señales para los operandos resultado del divisor.
signal malla0N00, malla0N01, malla0N02, malla0N03,
       malla0N10, malla0N11, malla0N12, malla0N13,
       malla0N20, malla0N21, malla0N22, malla0N23,
       malla0N30, malla0N31, malla0N32,
       malla0N33, malla1N00, malla1N01, malla1N02,
       malla1N03, malla1N10, malla1N11,
       malla1N12, malla1N13, malla1N20, malla1N21,
       malla1N22, malla1N23, malla1N30,
```

```
        malla1N31, malla1N32, malla1N33, malla2N00,
        malla2N01, malla2N02, malla2N03,
        malla2N10, malla2N11, malla2N12, malla2N13,
        malla2N20, malla2N21, malla2N22,
        malla2N23, malla2N30, malla2N31, malla2N32,
        malla2N33, malla3N00, malla3N01,
        malla3N02, malla3N03, malla3N10, malla3N11,
        malla3N12, malla3N13, malla3N20,
        malla3N21, malla3N22, malla3N23, malla3N30,
        malla3N31, malla3N32,
        malla3N33 : std_logic_vector(31 downto 0);

--Señales intermedias
signal puntoDes000, puntoDes200, puntoDes210,
        suma_201_200, puntoDes300, puntoDes320,
        suma_300_302, suma_201_300_302, puntoDes400,
        puntoDes310, puntoDes410, puntoDes420,
        puntoDes430, suma_403_400, suma_402_403_400,
        suma_301_302, suma_401_301_302,
        suma_402_403_400_401_301_302, puntoDes201,
        suma_200_210, puntoDes301, puntoDes311,
        suma_311_310, suma_300_301,
        suma_311_310_300_301, puntoDes401, puntoDes421,
        suma_412_311, suma_410_311_412, suma_400_301,
        suma_402_400_301, suma_410_412_311_402_400_301,
        puntoDes500, puntoDes501, puntoDes510,
        puntoDes411, puntoDes511, puntoDes520,
        puntoDes521, puntoDes530, puntoDes531,
        suma_510_511, suma_411_510_511, suma_513_412,
        suma_512_513_412, suma_411_510_511_512_513_412,
        suma_500_501, suma_402_500_501, suma_401_503,
        suma_502_503_401, suma_500_402_501_502_401_503,
        suma_411_510_511_512_513_412_402_500_501_502_
        503_401, puntoDes302, suma_300_210,
        suma_320_300_210, puntoDes402, puntoDes412,
        suma_310_400, suma_311_310_400, suma_421_420,
        suma_401_421_420, suma_311_310_400_401_421_420,
        puntoDes502, puntoDes522, suma_410_311,
        suma_412_500, suma_410_311_412_500,
        suma_502_401, suma_520_522, suma_421_520_522,
        suma_502_401_520_421_522, suma_410_412_311_500_
```

502_401_421_520_522 , puntoDes600 , puntoDes602 ,
puntoDes610 , puntoDes512 , puntoDes612 ,
puntoDes620 , puntoDes622 , puntoDes630 ,
puntoDes632 , suma_411_510 , suma_511_512 ,
suma_411_510_511_512 , suma_502_622 ,
suma_501_502_622 , suma_513_412_501_502_622 ,
suma_411_510_511_512_513_412_501_502_622 ,
suma_601_600 , suma_603_602 ,
suma_601_600_603_602 , suma_521_623 ,
suma_522_621 , suma_620_522_621 ,
suma_623_521_522_620_621 ,
suma_601_600_603_602_521_623_620_522_621 ,
suma_411_510_511_512_513_412_501_502_622_623_
521_522_620_621_601_600_603_602 ,
puntoDes403 , suma_320_430 , suma_410_320_430 ,
suma_400_420 , suma_310_400_420 ,
suma_410_320_430_310_400_420 , puntoDes503 ,
puntoDes513 , suma_410_411_510 , suma_531_530 ,
suma_511_531_530 , suma_410_411_510_511_531_530 ,
suma_501_520 , suma_500_501_520 , suma_420_521 ,
suma_421_521_420 , suma_500_501_520_421_420_521 ,
suma_410_411_510_511_531_530_500_501_421_520_
521_420 , puntoDes601 , puntoDes603 , puntoDes621 ,
puntoDes623 , suma_610_411 ,
suma_610_411_510_511 , suma_512_612 ,
suma_622_520 , suma_501_520_622 , suma_512_612_
501_622_520 , suma_610_411_510_511_512_612_501_
520_622 , suma_630_531 , suma_600_602 ,
suma_630_531_600_602 , suma_632_421 ,
suma_522_620 , suma_521_620_522 ,
suma_632_421_521_522_620 , suma_630_531_600_602_
632_421_521_620_522 , suma_610_411_510_511_512_
612_501_622_520_521_522_620_630_531_600_632_
602_421 , puntoDes700 , puntoDes701 , puntoDes702 ,
puntoDes703 , puntoDes710 , puntoDes711 ,
puntoDes712 , puntoDes613 , puntoDes713 ,
puntoDes720 , puntoDes721 , puntoDes422 ,
puntoDes722 , puntoDes723 , puntoDes730 ,
puntoDes631 , puntoDes731 , puntoDes732 ,
puntoDes733 , suma_713_411 , suma_610_713_411 ,
suma_712_412 , suma_613_623 ,

```
suma_712_412_613_623 , suma_713_610_411_712_
412_613_623 , suma_703_700 , suma_702_703_700 ,
suma_701_631 , suma_632_620 , suma_701_631_
632_620 , suma_702_703_700_701_631_632_620 ,
suma_610_713_712_411_412_613_623_620_702_703_
700_701_631_632 , suma_731_732 ,
suma_730_731_732 , suma_733_601 ,
suma_711_710 , suma_733_601_711_710 ,
suma_730_731_732_733_601_711_710 , suma_721_722 ,
suma_720_721_722 , suma_602_723 , suma_421_422 ,
suma_723_602_421_422 , suma_720_721_722_602_
723_421_422 , suma_711_710_730_731_732_733_601_
720_721_722_723_602_421_422 ,
suma_610_411_412_613_623_620_702_703_700_701_
631_632_713_712_711_710_730_731_732_733_601_
720_721_722_723_602_421_422 ,
puntoDes202 , puntoDes303 , suma_310_330 ,
suma_220_310_330 , puntoDes312 , puntoDes413 ,
suma_411_431 , suma_410_411_431 , suma_321_430 ,
suma_320_430_321 , suma_410_411_431_320_321_430 ,
puntoDes523 , suma_532_411 , suma_510_512 ,
suma_411_532_510_512 , suma_431_530 ,
suma_420_422 , suma_321_420_422 ,
suma_431_530_321_420_422 , suma_532_411_510_512_
431_321_530_420_422 , puntoDes611 , puntoDes532 ,
puntoDes633 , suma_610_611 ,
suma_610_611_511_512 , suma_612_613 ,
suma_521_522 , suma_520_521_522 ,
suma_612_613_520_521_522 , suma_610_611_511_512_
612_613_520_521_522 , suma_532_631 ,
suma_532_631_630_531 , suma_633_632 ,
suma_523_422 , suma_421_422_523 , suma_633_632_
421_523_422 , suma_532_631_630_531_633_632_421_
422_523 , suma_610_611_511_512_612_613_520_521_
522_523_532_631_630_531_633_632_421_422 ,
puntoDes203 , suma_220_230 , puntoDes313 ,
suma_320_321 , suma_330_331 ,
suma_320_321_330_331 , puntoDes423 ,
suma_321_432 , suma_430_321_432 , suma_331_422 ,
suma_420_331_422 , suma_321_430_432_331_420_422 ,
puntoDes533 , suma_532_431 , suma_533_532_431 ,
```

suma_530_432 , suma_531_530_432 ,
suma_533_532_431_531_530_432 , suma_520_521 ,
suma_421_520_521 , suma_522_523_422 ,
suma_520_421_521_522_422_523 , suma_533_532_531_
431_530_432_421_520_521_522_523_422 ,
puntoDes003 , puntoDes213 , suma_231_230 ,
puntoDes323 , suma_332_330 , suma_231_332_330 ,
puntoDes433 , suma_430_432 , suma_431_430_432 ,
suma_332_331 , suma_433_332_331 ,
suma_431_430_432_433_332_331 , puntoDes220 ,
puntoDes330 , suma_301_303 , suma_202_301_303 ,
puntoDes230 , suma_203_202 , puntoDes030 ,
puntoDes321 , puntoDes431 , suma_413_403 ,
suma_411_413_403 , suma_401_302 ,
suma_312_401_302 , suma_411_413_403_312_401_302 ,
puntoDes331 , suma_313_312 , suma_302_303 ,
suma_313_312_302_303 , puntoDes231 ,
suma_203_213 , suma_411_413 , suma_402_312 ,
suma_411_413_402_312 , suma_501_503 ,
suma_521_523_422 , suma_501_503_521_422_523 ,
suma_411_413_402_312_501_503_521_523_422 ,
puntoDes432 , suma_313_403 , suma_402_313_403 ,
suma_423_422 , suma_312_423_422 ,
suma_313_402_403_312_423_422 , puntoDes332 ,
suma_213_303 , suma_323_213_303 ,
suma_611_511 , suma_512_513 ,
suma_611_511_512_513 , suma_412_613 ,
suma_502_521_623 , suma_412_613_502_623_521 ,
suma_611_511_512_513_412_613_502_521_623 ,
suma_601_603 , suma_532_631_601_603 ,
suma_633_522 , suma_523_621 , suma_422_621_523 ,
suma_633_522_523_621_422 , suma_532_631_601_603_
633_522_422_621_523 , suma_611_511_512_513_412_
613_502_623_521_522_523_621_532_631_601_633_
603_422 , suma_532_512 , suma_533_532_512 ,
suma_412_413 , suma_513_412_413 ,
suma_533_532_512_513_412_413 , suma_503_522 ,
suma_502_503_522 , suma_423_523_422 ,
suma_502_503_522_423_422_523 ,
suma_533_532_512_513_412_413_502_503_423_522_
523_422 , suma_413_313 , suma_323_413_313 ,

```
        suma_433_423, suma_403_433_423, suma_323_413_
        313_433_403_423, puntoDes322, suma_533_511,
        suma_322_513, suma_533_511_322_513,
        suma_412_531, suma_421_423, suma_432_421_423,
        suma_531_412_432_421_423, suma_533_511_322_513_
        412_531_432_421_423, suma_323_412,
        suma_322_323_412, suma_432_433,
        suma_413_432_433, suma_322_323_412_413_432_433,
        puntoDes232, puntoDes333, suma_313_333,
        suma_223_313_333, suma_431_433,
        suma_322_431_433, suma_332_421_423,
        suma_322_431_433_332_421_423, suma_322_323,
        suma_332_333, suma_322_323_332_333,
        puntoDes233, suma_223_233, puntoDes223,
        suma_333_331, suma_232_333_331, suma_233_232,
        puntoDes033 : std_logic_vector(31 downto 0);

begin

-------------------------------
 ------- PROTOCOLO -------
-------------------------------
-- Entradas de control:
set <= entradaControl(17);
get <= entradaControl(18);
leido <= entradaControl(19);
escrito <= entradaControl(20);

-- Salidas de control:
salidaControl(0) <= datoOk;
salidaControl(1) <= rdy;
salidaControl(31 downto 2) <= (others => '0');

-- Datos
salidaDatos <= datosOut;
datosIn <= entradaDatos;

controlador:
process(clk, rst)
begin
        rdy <= '0';
```

```
datoOk <= '0';
eDecodificador <= '0';
eMultiplexor <= '0';
if (rst = '0') then
        estado <= espera;
        contador <= 0;
elsif (clk'event and clk = '1') then
        case estado is
           when espera =>
             contador <= 0;
             if(set = '1') then
                estado <= escribiendo;
              elsif(get = '1') then
                estado <= leyendo;
             end if;

           when escribiendo =>
             if(escrito = '1')then
                if(contador = 15) then
                  estado <= espera;
                else
                  estado <= direccionando;
                  contador <= contador + 1;
                end if;
             end if;

           when direccionando =>
             if(set='1') then
                estado<=escribiendo;
             elsif(get='1') then
                estado<=leyendo;
             end if;

            when leyendo =>
              if(leido = '1') then
                if(contador = 63) then
                   estado <= espera;
                else
                   estado <= direccionando;
                   contador <= contador + 1;
                end if;
```

```
                    end if;

                when others => null;
            end case;
        end if;

        case estado is
            when espera =>
                rdy <= '1';
            when escribiendo =>
                eDecodificador <= '1';
                datoOk <= '1';
            when leyendo =>
                eMultiplexor <= '1';
                datoOk <= '1';
            when direccionando =>
                null;
            when others =>
                null;
        end case;
end process controlador;

Decodificador:
process (clk,rst)
begin
        if(rst='0') then
            regE00 <= (others => '0');
            regE01 <= (others => '0');
            regE02 <= (others => '0');
            regE03 <= (others => '0');
            regE10 <= (others => '0');
            regE11 <= (others => '0');
            regE12 <= (others => '0');
            regE13 <= (others => '0');
            regE20 <= (others => '0');
            regE21 <= (others => '0');
            regE22 <= (others => '0');
            regE23 <= (others => '0');
            regE30 <= (others => '0');
            regE31 <= (others => '0');
            regE32 <= (others => '0');
```

```
                regE33 <= (others => '0');
        elsif(clk'event and clk='1') then
                if(eDecodificador = '1') then
                   case contador is
                        when 0 => regE00 <= datosIn;
                        when 1 => regE01 <= datosIn;
                        when 2 => regE02 <= datosIn;
                        when 3 => regE03 <= datosIn;
                        when 4 => regE10 <= datosIn;
                        when 5 => regE11 <= datosIn;
                        when 6 => regE12 <= datosIn;
                        when 7 => regE13 <= datosIn;
                        when 8 => regE20 <= datosIn;
                        when 9 => regE21 <= datosIn;
                        when 10 => regE22 <= datosIn;
                        when 11 => regE23 <= datosIn;
                        when 12 => regE30 <= datosIn;
                        when 13 => regE31 <= datosIn;
                        when 14 => regE32 <= datosIn;
                        when 15 => regE33 <= datosIn;
                        when others =>null;
                    end case;
                end if;
        end if;
end process;

multiplexor:
process(rst,clk)
begin
        datosOut <= (others => '0');
        if(eMultiplexor='1') then
            case contador is
                when 0 => datosOut <= malla0N00;
                when 1 => datosOut <= malla0N01;
                when 2 => datosOut <= malla0N02;
                when 3 => datosOut <= malla0N03;
                when 4 => datosOut <= malla0N10;
                when 5 => datosOut <= malla0N11;
                when 6 => datosOut <= malla0N12;
                when 7 => datosOut <= malla0N13;
                when 8 => datosOut <= malla0N20;
```

```
when 9 => datosOut <= malla0N21;
when 10 => datosOut <= malla0N22;
when 11 => datosOut <= malla0N23;
when 12 => datosOut <= malla0N30;
when 13 => datosOut <= malla0N31;
when 14 => datosOut <= malla0N32;
when 15 => datosOut <= malla0N33;
when 16 => datosOut <= malla1N00;
when 17 => datosOut <= malla1N01;
when 18 => datosOut <= malla1N02;
when 19 => datosOut <= malla1N03;
when 20 => datosOut <= malla1N10;
when 21 => datosOut <= malla1N11;
when 22 => datosOut <= malla1N12;
when 23 => datosOut <= malla1N13;
when 24 => datosOut <= malla1N20;
when 25 => datosOut <= malla1N21;
when 26 => datosOut <= malla1N22;
when 27 => datosOut <= malla1N23;
when 28 => datosOut <= malla1N30;
when 29 => datosOut <= malla1N31;
when 30 => datosOut <= malla1N32;
when 31 => datosOut <= malla1N33;
when 32 => datosOut <= malla2N00;
when 33 => datosOut <= malla2N01;
when 34 => datosOut <= malla2N02;
when 35 => datosOut <= malla2N03;
when 36 => datosOut <= malla2N10;
when 37 => datosOut <= malla2N11;
when 38 => datosOut <= malla2N12;
when 39 => datosOut <= malla2N13;
when 40 => datosOut <= malla2N20;
when 41 => datosOut <= malla2N21;
when 42 => datosOut <= malla2N22;
when 43 => datosOut <= malla2N23;
when 44 => datosOut <= malla2N30;
when 45 => datosOut <= malla2N31;
when 46 => datosOut <= malla2N32;
when 47 => datosOut <= malla2N33;
when 48 => datosOut <= malla3N00;
when 49 => datosOut <= malla3N01;
```

```
                when 50 => datosOut <= malla3N02;
                when 51 => datosOut <= malla3N03;
                when 52 => datosOut <= malla3N10;
                when 53 => datosOut <= malla3N11;
                when 54 => datosOut <= malla3N12;
                when 55 => datosOut <= malla3N13;
                when 56 => datosOut <= malla3N20;
                when 57 => datosOut <= malla3N21;
                when 58 => datosOut <= malla3N22;
                when 59 => datosOut <= malla3N23;
                when 60 => datosOut <= malla3N30;
                when 61 => datosOut <= malla3N31;
                when 62 => datosOut <= malla3N32;
                when 63 => datosOut <= malla3N33;
                when others =>
                        datosOut <= (others => '0');
            end case;
        end if;
end process;

-------------------------
------ DIVISOR ------
-------------------------

puntoDes000 <= regE00;
puntoDes200 <= "00" & regE00(31 downto 2);
puntoDes210 <= "00" & regE10(31 downto 2);
puntoDes300 <= "000" & regE00(31 downto 3);
puntoDes320 <= "000" & regE20(31 downto 3);
puntoDes400 <= "0000" & regE00(31 downto 4);
puntoDes310 <= "000" & regE10(31 downto 3);
puntoDes410 <= "0000" & regE10(31 downto 4);
puntoDes420 <= "0000" & regE20(31 downto 4);
puntoDes430 <= "0000" & regE30(31 downto 4);
puntoDes201 <= "00" & regE01(31 downto 2);
puntoDes301 <= "000" & regE01(31 downto 3);
puntoDes311 <= "000" & regE11(31 downto 3);
puntoDes401 <= "0000" & regE01(31 downto 4);
puntoDes421 <= "0000" & regE21(31 downto 4);
puntoDes500 <= "00000" & regE00(31 downto 5);
puntoDes501 <= "00000" & regE01(31 downto 5);
```

```
puntoDes510 <= "00000" & regE10(31 downto 5);
puntoDes411 <= "0000" & regE11(31 downto 4);
puntoDes511 <= "00000" & regE11(31 downto 5);
puntoDes520 <= "00000" & regE20(31 downto 5);
puntoDes521 <= "00000" & regE21(31 downto 5);
puntoDes530 <= "00000" & regE30(31 downto 5);
puntoDes531 <= "00000" & regE31(31 downto 5);
puntoDes302 <= "000" & regE02(31 downto 3);
puntoDes402 <= "0000" & regE02(31 downto 4);
puntoDes412 <= "0000" & regE12(31 downto 4);
puntoDes502 <= "00000" & regE02(31 downto 5);
puntoDes522 <= "00000" & regE22(31 downto 5);
puntoDes600 <= "000000" & regE00(31 downto 6);
puntoDes602 <= "000000" & regE02(31 downto 6);
puntoDes610 <= "000000" & regE10(31 downto 6);
puntoDes512 <= "00000" & regE12(31 downto 5);
puntoDes612 <= "000000" & regE12(31 downto 6);
puntoDes620 <= "000000" & regE20(31 downto 6);
puntoDes622 <= "000000" & regE22(31 downto 6);
puntoDes630 <= "000000" & regE30(31 downto 6);
puntoDes632 <= "000000" & regE32(31 downto 6);
puntoDes403 <= "0000" & regE03(31 downto 4);
puntoDes503 <= "00000" & regE03(31 downto 5);
puntoDes513 <= "00000" & regE13(31 downto 5);
puntoDes601 <= "000000" & regE01(31 downto 6);
puntoDes603 <= "000000" & regE03(31 downto 6);
puntoDes621 <= "000000" & regE21(31 downto 6);
puntoDes623 <= "000000" & regE23(31 downto 6);
puntoDes700 <= "0000000" & regE00(31 downto 7);
puntoDes701 <= "0000000" & regE01(31 downto 7);
puntoDes702 <= "0000000" & regE02(31 downto 7);
puntoDes703 <= "0000000" & regE03(31 downto 7);
puntoDes710 <= "0000000" & regE10(31 downto 7);
puntoDes711 <= "0000000" & regE11(31 downto 7);
puntoDes712 <= "0000000" & regE12(31 downto 7);
puntoDes613 <= "000000" & regE13(31 downto 6);
puntoDes713 <= "0000000" & regE13(31 downto 7);
puntoDes720 <= "0000000" & regE20(31 downto 7);
puntoDes721 <= "0000000" & regE21(31 downto 7);
puntoDes422 <= "0000" & regE22(31 downto 4);
puntoDes722 <= "0000000" & regE22(31 downto 7);
```

```
puntoDes723 <= "0000000" &  regE23(31 downto 7);
puntoDes730 <= "0000000" &  regE30(31 downto 7);
puntoDes631 <= "000000" &  regE31(31 downto 6);
puntoDes731 <= "0000000" &  regE31(31 downto 7);
puntoDes732 <= "0000000" &  regE32(31 downto 7);
puntoDes733 <= "0000000" &  regE33(31 downto 7);
puntoDes202 <= "00" &  regE02(31 downto 2);
puntoDes303 <= "000" &  regE03(31 downto 3);
puntoDes312 <= "000" &  regE12(31 downto 3);
puntoDes413 <= "0000" &  regE13(31 downto 4);
puntoDes523 <= "00000" &  regE23(31 downto 5);
puntoDes611 <= "000000" &  regE11(31 downto 6);
puntoDes532 <= "00000" &  regE32(31 downto 5);
puntoDes633 <= "000000" &  regE33(31 downto 6);
puntoDes203 <= "00" &  regE03(31 downto 2);
puntoDes313 <= "000" &  regE13(31 downto 3);
puntoDes423 <= "0000" &  regE23(31 downto 4);
puntoDes533 <= "00000" &  regE33(31 downto 5);
puntoDes003 <= regE03;
puntoDes213 <= "00" &  regE13(31 downto 2);
puntoDes323 <= "000" &  regE23(31 downto 3);
puntoDes433 <= "0000" &  regE33(31 downto 4);
puntoDes220 <= "00" &  regE20(31 downto 2);
puntoDes330 <= "000" &  regE30(31 downto 3);
puntoDes230 <= "00" &  regE30(31 downto 2);
puntoDes030 <= regE30;
puntoDes321 <= "000" &  regE21(31 downto 3);
puntoDes431 <= "0000" &  regE31(31 downto 4);
puntoDes331 <= "000" &  regE31(31 downto 3);
puntoDes231 <= "00" &  regE31(31 downto 2);
puntoDes432 <= "0000" &  regE32(31 downto 4);
puntoDes332 <= "000" &  regE32(31 downto 3);
puntoDes322 <= "000" &  regE22(31 downto 3);
puntoDes232 <= "00" &  regE32(31 downto 2);
puntoDes333 <= "000" &  regE33(31 downto 3);
puntoDes233 <= "00" &  regE33(31 downto 2);
puntoDes223 <= "00" &  regE23(31 downto 2);
puntoDes033 <= regE33;

suma_201_200 <= puntoDes210 + puntoDes200;
suma_300_302 <= puntoDes300 + puntoDes320;
```

```
suma_201_300_302 <= puntoDes210 + suma_300_302;
suma_403_400 <= puntoDes430 + puntoDes400;
suma_402_403_400 <= puntoDes420 + suma_403_400;
suma_301_302 <= puntoDes310 + puntoDes320;
suma_401_301_302 <= puntoDes410 + suma_301_302;
suma_402_403_400_401_301_302 <= suma_402_403_400 +
suma_401_301_302;
suma_200_210 <= puntoDes200 + puntoDes201;
suma_311_310 <= puntoDes311 + puntoDes301;
suma_300_301 <= puntoDes300 + puntoDes310;
suma_311_310_300_301 <= suma_311_310 + suma_300_301;
suma_412_311 <= puntoDes421 + puntoDes311;
suma_410_311_412 <= puntoDes401 + suma_412_311;
suma_400_301 <= puntoDes400 + puntoDes310;
suma_402_400_301 <= puntoDes420 + suma_400_301;
suma_410_412_311_402_400_301 <= suma_410_311_412 +
suma_402_400_301;
suma_510_511 <= puntoDes501 + puntoDes511;
suma_411_510_511 <= puntoDes411 + suma_510_511;
suma_513_412 <= puntoDes531 + puntoDes421;
suma_512_513_412 <= puntoDes521 + suma_513_412;
suma_411_510_511_512_513_412 <= suma_411_510_511 +
suma_512_513_412;
suma_500_501 <= puntoDes500 + puntoDes510;
suma_402_500_501 <= puntoDes420 + suma_500_501;
suma_401_503 <= puntoDes410 + puntoDes530;
suma_502_503_401 <= puntoDes520 + suma_401_503;
suma_500_402_501_502_401_503 <= suma_402_500_501 +
suma_502_503_401;
suma_411_510_511_512_513_412_402_500_501_502_503_401 <=
 suma_411_510_511_512_513_412 + suma_500_402_501_502_
401_503;
suma_300_210 <= puntoDes300 + puntoDes201;
suma_320_300_210 <= puntoDes302 + suma_300_210;
suma_310_400 <= puntoDes301 + puntoDes400;
suma_311_310_400 <= puntoDes311 + suma_310_400;
suma_421_420 <= puntoDes412 + puntoDes402;
suma_401_421_420 <= puntoDes410 + suma_421_420;
suma_311_310_400_401_421_420 <= suma_311_310_400 +
suma_401_421_420;
suma_410_311 <= puntoDes401 + puntoDes311;
```

```
suma_412_500 <= puntoDes421 + puntoDes500;
suma_410_311_412_500 <= suma_410_311 + suma_412_500;
suma_502_401 <= puntoDes520 + puntoDes410;
suma_520_522 <= puntoDes502 + puntoDes522;
suma_421_520_522 <= puntoDes412 + suma_520_522;
suma_502_401_520_421_522 <= suma_502_401 +
suma_421_520_522;
suma_410_412_311_500_502_401_421_520_522 <=
 suma_410_311_412_500 + suma_502_401_520_421_522;
suma_411_510 <= puntoDes411 + puntoDes501;
suma_511_512 <= puntoDes511 + puntoDes521;
suma_411_510_511_512 <= suma_411_510 + suma_511_512;
suma_502_622 <= puntoDes520 + puntoDes622;
suma_501_502_622 <= puntoDes510 + suma_502_622;
suma_513_412_501_502_622 <= suma_513_412 +
suma_501_502_622;
suma_411_510_511_512_513_412_501_502_622 <=
 suma_411_510_511_512 + suma_513_412_501_502_622;
suma_601_600 <= puntoDes610 + puntoDes600;
suma_603_602 <= puntoDes630 + puntoDes620;
suma_601_600_603_602 <= suma_601_600 + suma_603_602;
suma_521_623 <= puntoDes512 + puntoDes632;
suma_522_621 <= puntoDes522 + puntoDes612;
suma_620_522_621 <= puntoDes602 + suma_522_621;
suma_623_521_522_620_621 <= suma_521_623 +
suma_620_522_621;
suma_601_600_603_602_521_623_620_522_621 <=
 suma_601_600_603_602 + suma_623_521_522_620_621;
suma_411_510_511_512_513_412_501_502_622_623_521_522_
620_621_601_600_603_602 <= suma_411_510_511_512_513_
412_501_502_622 + suma_601_600_603_602_521_623_
620_522_621;
suma_320_430 <= puntoDes302 + puntoDes403;
suma_410_320_430 <= puntoDes401 + suma_320_430;
suma_400_420 <= puntoDes400 + puntoDes402;
suma_310_400_420 <= puntoDes301 + suma_400_420;
suma_410_320_430_310_400_420 <= suma_410_320_430 +
suma_310_400_420;
suma_410_411_510 <= puntoDes401 + suma_411_510;
suma_531_530 <= puntoDes513 + puntoDes503;
suma_511_531_530 <= puntoDes511 + suma_531_530;
```

```
suma_410_411_510_511_531_530 <= suma_410_411_510 +
suma_511_531_530;
suma_501_520 <= puntoDes510 + puntoDes502;
suma_500_501_520 <= puntoDes500 + suma_501_520;
suma_420_521 <= puntoDes402 + puntoDes512;
suma_421_521_420 <= puntoDes412 + suma_420_521;
suma_500_501_520_421_420_521 <= suma_500_501_520 +
suma_421_521_420;
suma_410_411_510_511_531_530_500_501_421_520_521_420 <=
 suma_410_411_510_511_531_530 + suma_500_501_520_421_
420_521;
suma_610_411 <= puntoDes601 + puntoDes411;
suma_610_411_510_511 <= suma_610_411 + suma_510_511;
suma_512_612 <= puntoDes521 + puntoDes621;
suma_622_520 <= puntoDes622 + puntoDes502;
suma_501_520_622 <= puntoDes510 + suma_622_520;
suma_512_612_501_622_520 <= suma_512_612 +
suma_501_520_622;
suma_610_411_510_511_512_612_501_520_622 <=
 suma_610_411_510_511 + suma_512_612_501_622_520;
suma_630_531 <= puntoDes603 + puntoDes513;
suma_600_602 <= puntoDes600 + puntoDes620;
suma_630_531_600_602 <= suma_630_531 + suma_600_602;
suma_632_421 <= puntoDes623 + puntoDes412;
suma_522_620 <= puntoDes522 + puntoDes602;
suma_521_620_522 <= puntoDes512 + suma_522_620;
suma_632_421_521_522_620 <= suma_632_421 +
suma_521_620_522;
suma_630_531_600_602_632_421_521_620_522 <=
 suma_630_531_600_602 + suma_632_421_521_522_620;
suma_610_411_510_511_512_612_501_622_520_521_522_620_
630_531_600_632_602_421 <= suma_610_411_510_511_512_
612_501_520_622 + suma_630_531_600_602_632_421_
521_620_522;
suma_713_411 <= puntoDes731 + puntoDes411;
suma_610_713_411 <= puntoDes601 + suma_713_411;
suma_712_412 <= puntoDes721 + puntoDes421;
suma_613_623 <= puntoDes631 + puntoDes632;
suma_712_412_613_623 <= suma_712_412 + suma_613_623;
suma_713_610_411_712_412_613_623 <= suma_610_713_411 +
suma_712_412_613_623;
```

```
suma_703_700 <= puntoDes730 + puntoDes700;
suma_702_703_700 <= puntoDes720 + suma_703_700;
suma_701_631 <= puntoDes710 + puntoDes613;
suma_632_620 <= puntoDes623 + puntoDes602;
suma_701_631_632_620 <= suma_701_631 + suma_632_620;
suma_702_703_700_701_631_632_620 <= suma_702_703_700 +
suma_701_631_632_620;
suma_610_713_712_411_412_613_623_620_702_703_700_701_
631_632 <= suma_713_610_411_712_412_613_623 +
suma_702_703_700_701_631_632_620;
suma_731_732 <= puntoDes713 + puntoDes723;
suma_730_731_732 <= puntoDes703 + suma_731_732;
suma_733_601 <= puntoDes733 + puntoDes610;
suma_711_710 <= puntoDes711 + puntoDes701;
suma_733_601_711_710 <= suma_733_601 + suma_711_710;
suma_730_731_732_733_601_711_710 <= suma_730_731_732 +
suma_733_601_711_710;
suma_721_722 <= puntoDes712 + puntoDes722;
suma_720_721_722 <= puntoDes702 + suma_721_722;
suma_602_723 <= puntoDes620 + puntoDes732;
suma_421_422 <= puntoDes412 + puntoDes422;
suma_723_602_421_422 <= suma_602_723 + suma_421_422;
suma_720_721_722_602_723_421_422 <= suma_720_721_722 +
suma_723_602_421_422;
suma_711_710_730_731_732_733_601_720_721_722_723_602_
421_422 <= suma_730_731_732_733_601_711_710 +
suma_720_721_722_602_723_421_422;
suma_610_411_412_613_623_620_702_703_700_701_631_632_
713_712_711_710_730_731_732_733_601_720_721_722_723_
602_421_422 <= suma_610_713_712_411_412_613_623_620_
702_703_700_701_631_632 + suma_711_710_730_731_732_
733_601_720_721_722_723_602_421_422;
suma_310_330 <= puntoDes301 + puntoDes303;
suma_220_310_330 <= puntoDes202 + suma_310_330;
suma_411_431 <= puntoDes411 + puntoDes413;
suma_410_411_431 <= puntoDes401 + suma_411_431;
suma_321_430 <= puntoDes312 + puntoDes403;
suma_320_430_321 <= puntoDes302 + suma_321_430;
suma_410_411_431_320_321_430 <= suma_410_411_431 +
suma_320_430_321;
suma_532_411 <= puntoDes523 + puntoDes411;
```

```
suma_510_512 <= puntoDes501 + puntoDes521;
suma_411_532_510_512 <= suma_532_411 + suma_510_512;
suma_431_530 <= puntoDes413 + puntoDes503;
suma_420_422 <= puntoDes402 + puntoDes422;
suma_321_420_422 <= puntoDes312 + suma_420_422;
suma_431_530_321_420_422 <= suma_431_530 +
suma_321_420_422;
suma_532_411_510_512_431_321_530_420_422 <=
 suma_411_532_510_512 + suma_431_530_321_420_422;
suma_610_611 <= puntoDes601 + puntoDes611;
suma_610_611_511_512 <= suma_610_611 + suma_511_512;
suma_612_613 <= puntoDes621 + puntoDes631;
suma_521_522 <= puntoDes512 + puntoDes522;
suma_520_521_522 <= puntoDes502 + suma_521_522;
suma_612_613_520_521_522 <= suma_612_613 +
suma_520_521_522;
suma_610_611_511_512_612_613_520_521_522 <=
 suma_610_611_511_512 + suma_612_613_520_521_522;
suma_532_631 <= puntoDes523 + puntoDes613;
suma_532_631_630_531 <= suma_532_631 + suma_630_531;
suma_633_632 <= puntoDes633 + puntoDes623;
suma_523_422 <= puntoDes532 + puntoDes422;
suma_421_422_523 <= puntoDes412 + suma_523_422;
suma_633_632_421_523_422 <= suma_633_632 +
suma_421_422_523;
suma_532_631_630_531_633_632_421_422_523 <=
suma_532_631_630_531 + suma_633_632_421_523_422;
suma_610_611_511_512_612_613_520_521_522_523_532_
631_630_531_633_632_421_422 <= suma_610_611_511_
512_612_613_520_521_522 + suma_532_631_630_531_633_
632_421_422_523;
suma_220_230 <= puntoDes202 + puntoDes203;
suma_320_321 <= puntoDes302 + puntoDes312;
suma_330_331 <= puntoDes303 + puntoDes313;
suma_320_321_330_331 <= suma_320_321 + suma_330_331;
suma_321_432 <= puntoDes312 + puntoDes423;
suma_430_321_432 <= puntoDes403 + suma_321_432;
suma_331_422 <= puntoDes313 + puntoDes422;
suma_420_331_422 <= puntoDes402 + suma_331_422;
suma_321_430_432_331_420_422 <= suma_430_321_432 +
suma_420_331_422;
```

```
suma_532_431 <= puntoDes523 + puntoDes413;
suma_533_532_431 <= puntoDes533 + suma_532_431;
suma_530_432 <= puntoDes503 + puntoDes423;
suma_531_530_432 <= puntoDes513 + suma_530_432;
suma_533_532_431_531_530_432 <= suma_533_532_431 +
suma_531_530_432;
suma_520_521 <= puntoDes502 + puntoDes512;
suma_421_520_521 <= puntoDes412 + suma_520_521;
suma_522_523_422 <= puntoDes522 + suma_523_422;
suma_520_421_521_522_422_523 <= suma_421_520_521 +
suma_522_523_422;
suma_533_532_531_431_530_432_421_520_521_522_523_422 <=
 suma_533_532_431_531_530_432 + suma_520_421_521_522_
422_523;
suma_231_230 <= puntoDes213 + puntoDes203;
suma_332_330 <= puntoDes323 + puntoDes303;
suma_231_332_330 <= puntoDes213 + suma_332_330;
suma_430_432 <= puntoDes403 + puntoDes423;
suma_431_430_432 <= puntoDes413 + suma_430_432;
suma_332_331 <= puntoDes323 + puntoDes313;
suma_433_332_331 <= puntoDes433 + suma_332_331;
suma_431_430_432_433_332_331 <= suma_431_430_432 +
suma_433_332_331;
suma_301_303 <= puntoDes310 + puntoDes330;
suma_202_301_303 <= puntoDes220 + suma_301_303;
suma_203_202 <= puntoDes230 + puntoDes220;
suma_413_403 <= puntoDes431 + puntoDes430;
suma_411_413_403 <= puntoDes411 + suma_413_403;
suma_401_302 <= puntoDes410 + puntoDes320;
suma_312_401_302 <= puntoDes321 + suma_401_302;
suma_411_413_403_312_401_302 <= suma_411_413_403 +
suma_312_401_302;
suma_313_312 <= puntoDes331 + puntoDes321;
suma_302_303 <= puntoDes320 + puntoDes330;
suma_313_312_302_303 <= suma_313_312 + suma_302_303;
suma_203_213 <= puntoDes230 + puntoDes231;
suma_411_413 <= puntoDes411 + puntoDes431;
suma_402_312 <= puntoDes420 + puntoDes321;
suma_411_413_402_312 <= suma_411_413 + suma_402_312;
suma_501_503 <= puntoDes510 + puntoDes530;
suma_521_523_422 <= puntoDes512 + suma_523_422;
```

```
suma_501_503_521_422_523 <= suma_501_503 +
suma_521_523_422;
suma_411_413_402_312_501_503_521_523_422 <=
 suma_411_413_402_312 + suma_501_503_521_422_523;
suma_313_403 <= puntoDes331 + puntoDes430;
suma_402_313_403 <= puntoDes420 + suma_313_403;
suma_423_422 <= puntoDes432 + puntoDes422;
suma_312_423_422 <= puntoDes321 + suma_423_422;
suma_313_402_403_312_423_422 <= suma_402_313_403 +
 suma_312_423_422;
suma_213_303 <= puntoDes231 + puntoDes330;
suma_323_213_303 <= puntoDes332 + suma_213_303;
suma_611_511 <= puntoDes611 + puntoDes511;
suma_512_513 <= puntoDes521 + puntoDes531;
suma_611_511_512_513 <= suma_611_511 + suma_512_513;
suma_412_613 <= puntoDes421 + puntoDes631;
suma_502_521_623 <= puntoDes520 + suma_521_623;
suma_412_613_502_623_521 <= suma_412_613 +
suma_502_521_623;
suma_611_511_512_513_412_613_502_521_623 <=
 suma_611_511_512_513 + suma_412_613_502_623_521;
suma_601_603 <= puntoDes610 + puntoDes630;
suma_532_631_601_603 <= suma_532_631 + suma_601_603;
suma_633_522 <= puntoDes633 + puntoDes522;
suma_523_621 <= puntoDes532 + puntoDes612;
suma_422_621_523 <= puntoDes422 + suma_523_621;
suma_633_522_523_621_422 <= suma_633_522 +
suma_422_621_523;
suma_532_631_601_603_633_522_422_621_523 <=
 suma_532_631_601_603 + suma_633_522_523_621_422;
suma_611_511_512_513_412_613_502_623_521_522_523_
621_532_631_601_633_603_422 <= suma_611_511_512_
513_412_613_502_521_623 + suma_532_631_601_603_
633_522_422_621_523;
suma_532_512 <= puntoDes523 + puntoDes521;
suma_533_532_512 <= puntoDes533 + suma_532_512;
suma_412_413 <= puntoDes421 + puntoDes431;
suma_513_412_413 <= puntoDes531 + suma_412_413;
suma_533_532_512_513_412_413 <= suma_533_532_512 +
suma_513_412_413;
suma_503_522 <= puntoDes530 + puntoDes522;
```

```
suma_502_503_522 <= puntoDes520 + suma_503_522;
suma_423_523_422 <= puntoDes432 + suma_523_422;
suma_502_503_522_423_422_523 <= suma_502_503_522 +
suma_423_523_422;
suma_533_532_512_513_412_413_502_503_423_522_523_422 <=
 suma_533_532_512_513_412_413 +
suma_502_503_522_423_422_523;
suma_413_313 <= puntoDes431 + puntoDes331;
suma_323_413_313 <= puntoDes332 + suma_413_313;
suma_433_423 <= puntoDes433 + puntoDes432;
suma_403_433_423 <= puntoDes430 + suma_433_423;
suma_323_413_313_433_403_423 <= suma_323_413_313 +
suma_403_433_423;
suma_533_511 <= puntoDes533 + puntoDes511;
suma_322_513 <= puntoDes322 + puntoDes531;
suma_533_511_322_513 <= suma_533_511 + suma_322_513;
suma_412_531 <= puntoDes421 + puntoDes513;
suma_421_423 <= puntoDes412 + puntoDes432;
suma_432_421_423 <= puntoDes423 + suma_421_423;
suma_531_412_432_421_423 <= suma_412_531 +
suma_432_421_423;
suma_533_511_322_513_412_531_432_421_423 <=
 suma_533_511_322_513 + suma_531_412_432_421_423;
suma_323_412 <= puntoDes332 + puntoDes421;
suma_322_323_412 <= puntoDes322 + suma_323_412;
suma_432_433 <= puntoDes423 + puntoDes433;
suma_413_432_433 <= puntoDes431 + suma_432_433;
suma_322_323_412_413_432_433 <= suma_322_323_412 +
suma_413_432_433;
suma_313_333 <= puntoDes331 + puntoDes333;
suma_223_313_333 <= puntoDes232 + suma_313_333;
suma_431_433 <= puntoDes413 + puntoDes433;
suma_322_431_433 <= puntoDes322 + suma_431_433;
suma_332_421_423 <= puntoDes323 + suma_421_423;
suma_322_431_433_332_421_423 <= suma_322_431_433 +
suma_332_421_423;
suma_322_323 <= puntoDes322 + puntoDes332;
suma_332_333 <= puntoDes323 + puntoDes333;
suma_322_323_332_333 <= suma_322_323 + suma_332_333;
suma_223_233 <= puntoDes232 + puntoDes233;
suma_333_331 <= puntoDes333 + puntoDes313;
```

```
suma_232_333_331 <= puntoDes223 + suma_333_331;
suma_233_232 <= puntoDes233 + puntoDes223;

malla0N00 <= puntoDes000;
malla0N01 <= suma_201_200;
malla0N02 <= suma_201_300_302;
malla0N03 <= suma_402_403_400_401_301_302;
malla0N10 <= suma_200_210;
malla0N11 <= suma_311_310_300_301;
malla0N12 <= suma_410_412_311_402_400_301;
malla0N13 <= suma_411_510_511_512_513_412_402_500_501
_502_503_401;
malla0N20 <= suma_320_300_210;
malla0N21 <= suma_311_310_400_401_421_420;
malla0N22 <= suma_410_412_311_500_502_401_421_520_522;
malla0N23 <= suma_411_510_511_512_513_412_501_502_622_
623_521_522_620_621_601_600_603_602;
malla0N30 <= suma_410_320_430_310_400_420;
malla0N31 <= suma_410_411_510_511_531_530_500_501_421_
520_521_420;
malla0N32 <= suma_610_411_510_511_512_612_501_622_520_
521_522_620_630_531_600_632_602_421;
malla0N33 <= suma_610_411_412_613_623_620_702_703_700_7
01_631_632_713_712_711_710_730_731_732_733_601_720_721_
722_723_602_421_422;
malla1N00 <= suma_410_320_430_310_400_420;
malla1N01 <= suma_410_411_510_511_531_530_500_501_421_
520_521_420;
malla1N02 <= suma_610_411_510_511_512_612_501_622_
520_521_522_620_630_531_600_632_602_421;
malla1N03 <= suma_610_411_412_613_623_620_702_703_700_
701_631_632_713_712_711_710_730_731_732_733_601_720_
721_722_723_602_421_422;
malla1N10 <= suma_220_310_330;
malla1N11 <= suma_410_411_431_320_321_430;
malla1N12 <= suma_532_411_510_512_431_321_530_420_422;
malla1N13 <= suma_610_611_511_512_612_613_520_521_522_
523_532_631_630_531_633_632_421_422;
malla1N20 <= suma_220_230;
malla1N21 <= suma_320_321_330_331;
malla1N22 <= suma_321_430_432_331_420_422;
```

```
malla1N23 <= suma_533_532_531_431_530_432_421_520_521_
522_523_422;
malla1N30 <= puntoDes003;
malla1N31 <= suma_231_230;
malla1N32 <= suma_231_332_330;
malla1N33 <= suma_431_430_432_433_332_331;
malla2N00 <= suma_402_403_400_401_301_302;
malla2N01 <= suma_202_301_303;
malla2N02 <= suma_203_202;
malla2N03 <= puntoDes030;
malla2N10 <= suma_411_510_511_512_513_412_402_500_501_
502_503_401;
malla2N11 <= suma_411_413_403_312_401_302;
malla2N12 <= suma_313_312_302_303;
malla2N13 <= suma_203_213;
malla2N20 <= suma_411_510_511_512_513_412_501_502_622_
623_521_522_620_621_601_600_603_602;
malla2N21 <= suma_411_413_402_312_501_503_521_523_422;
malla2N22 <= suma_313_402_403_312_423_422;
malla2N23 <= suma_323_213_303;
malla2N30 <= suma_610_411_412_613_623_620_702_703_700_
701_631_632_713_712_711_710_730_731_732_733_601_720_
721_722_723_602_421_422;
malla2N31 <= suma_611_511_512_513_412_613_502_623_521_
522_523_621_532_631_601_633_603_422;
malla2N32 <= suma_533_532_512_513_412_413_502_503_423_
522_523_422;
malla2N33 <= suma_323_413_313_433_403_423;
malla3N00 <= suma_610_411_412_613_623_620_702_703_700_
701_631_632_713_712_711_710_730_731_732_733_601_720_
721_722_723_602_421_422;
malla3N01 <= suma_611_511_512_513_412_613_502_623_521_
522_523_621_532_631_601_633_603_422;
malla3N02 <= suma_533_532_512_513_412_413_502_503_423_
522_523_422;
malla3N03 <= suma_323_413_313_433_403_423;
malla3N10 <= suma_610_611_511_512_612_613_520_521_522_
523_532_631_630_531_633_632_421_422;
malla3N11 <= suma_533_511_322_513_412_531_432_421_423;
malla3N12 <= suma_322_323_412_413_432_433;
malla3N13 <= suma_223_313_333;
```

```
malla3N20 <= suma_533_532_531_431_530_432_421_520_521_
522_523_422;
malla3N21 <= suma_322_431_433_332_421_423;
malla3N22 <= suma_322_323_332_333;
malla3N23 <= suma_223_233;
malla3N30 <= suma_431_430_432_433_332_331;
malla3N31 <= suma_232_333_331;
malla3N32 <= suma_233_232;
malla3N33 <= puntoDes033;

end imp;
```

Apéndice E

Perfil del programa en C++

Para ayudarnos a decidir qué parte del algoritmo implementaríamos directamente en hardware, hicimos un perfil del programa implementado en C++ utilizando *gprof*[1]. Dicho programa funciona del siguiente modo:

1. Se compila el software con el compilador *gcc* y la opción `-pg`.

2. Se ejecuta el programa normalmente, con lo que éste genera un archivo `.out`.

3. Se ejecuta `gprof ejecutable archivo.out`, con lo que se imprime por salida estándar un informe del programa.

E.1. Perfil del programa original

Tras proceder como se indica en los pasos anteriores, obtuvimos un perfil, para una imagen de 800×600 pixels que resumimos a continuación:

Tiempo de ejecución (%)	**Llamadas**	**Función**
50.09	19.571.264	`nodoIntermedio`
13.43	10.813.904	`corta(Caja)`
9.41	1.223.205	`Caja(Punto***)` (Constructor de `caja`)

[1] `http://www.gnu.org`

E.2. Perfil en punto fijo

Tras analizar el perfil anterior y decidirnos por una implementación hardware, decidimos realizar un nuevo perfil, pero esta vez utilizando en el código las operaciones aritméticas en punto fijo. El perfil obtenido, resumido, fue el siguiente:

Tiempo de ejecución (%)	**Llamadas**	**Función**
33.60	20.585.024	`nodoIntermedio`
19.41	514.625.654	`prod_fijo`
11.26	66.289.876	Conversión de punto flotante a punto fijo.
8.21	74.756.970	Conversión de punto fijo a punto flotante.
7.18	10.897.780	`corta(Caja)`
5.00	1.286.565	`Caja(Punto***)` (Constructor de `caja`)

E.3. Conclusión

Vemos que al calcular en punto fijo los tiempos de ejecución de las distintas funciones cambian. Para empezar, como la precisión que se obtiene con punto fijo es distinta de la que se obtiene con punto flotante, la determinación de cuándo una caja es suficientemente pequeña cambia, por lo que el número de divisiones que se hace en uno y otro caso también cambia. Además, aunque en modo software se dedique casi un 20 % del tiempo a realizar productos en punto fijo, esos tiempos literalmente desaparecen al trabajar en hardware, porque son productos que se realizan combinacionalmente.

Además, gran parte del tiempo de ejecución se emplea en la conversión de punto flotante a punto fijo y viceversa, por lo que sería conveniente o bien trabajar en punto flotante todo el tiempo, o tener una técnica para convertir de punto flotante a punto fijo "gratis" —como, por ejemplo, utilizando un conversor hardware—.

Apéndice F

Código C++ para la FPGA

F.1. Acceso a VGA

Para aprender a mostrar imágenes en la pantalla conectada a la placa de desarrollo, creamos un programa de ejemplo que nos permitiese mostrar unas bandas de colores en el monitor. Su código es el siguiente:

```
#include "xparameters.h"
#include "xutil.h"

void testRGB(int  pixels);

int main (void) {
    print("-- Comenzando main() --\r\n");

    //Ejecutamos el ejemplo de VGA
    testRGB(309760);

    print("-- Terminando main() --\r\n");
    return 0;
}

void testRGB(int  pixels) {
    unsigned int x, num;
    volatile unsigned int color;
    volatile unsigned int  *pDisplay;

    //Inicializamos la dirección de escritura del
    //framebuffer
```

```
    pDisplay    = (volatile unsigned int  *)0x07E00000;
    x = num = 0;

    // Escribimos los colores en el framebuffer
    while (num < pixels) {
        // Patrón
        if (x < 80)
            color = 0x00FFFFFF; // Blanco
        else if (x < 160)
            color = 0x00000000; // Negro
        else if (x < 240)
            color = 0x00FFFF00; // Amarillo
        else if (x < 320)
            color = 0x0000FFFF; // Celeste
        else if (x < 400)
            color = 0x0000FF00; // Verde
        else if (x < 480)
            color = 0x00FF00FF; // Magenta
        else if (x < 560)
            color = 0x00FF0000; // Rojo
        else if (x < 640)
            color = 0x000000FF; // azul

        *(pDisplay) = color;
        pDisplay++;
        num++;
        x++;
        if (x >= 640) {
            x = 0;
            pDisplay += (1024 - 640);
        }
    }
}
```

F.2. Acceso a GPIOs

Para aprender a utilizar los pines de GPIO, creamos un sistema con *leds* y *switches*, y desarrollamos el siguiente programa, capaz de mostrar el valor de los *switches* en los *leds*.

```
#include "xparameters.h"
#include "xcache_l.h"
#include "stdio.h"
#include "xgpio.h"

int main(void) {
    //Declaración de los dispositivos GPIO
    XGpio leds;
    XGpio switches;
    unsigned int valor;

    print("Comenzando main()...\r\n");

    // Inicializamos GPIOs
    XGpio_Initialize(&leds, XPAR_GPIO_LED_ID);
    XGpio_Initialize(&switches, XPAR_GPIO_SWITCH_ID);
    print("GPIO inicializado\r\n");

    // Ajustamos las direcciones de los buffer
    // tri-estado
    XGpio_SetDataDirection(&leds, LED_CHANNEL, 0x0);
    XGpio_SetDataDirection(&switches, SWITCH_CHANNEL,
                           0xffffffff);

    // Leemos el valor de los switches y lo colocamos
    // en los leds
    while (1) {
        // Leemos el valor
        valor = XGpio_DiscreteRead(&switches,
                                   SWITCH_CHANNEL);

        // Volcamos el valor
        XGpio_DiscreteWrite(&leds, LED_CHANNEL, valor);
    }

    return 0;
}
```

F.3. Medición de tiempos

F.3.1. Programa principal

```
#include "xparameters.h"
#include "stdio.h"
#include "xtime_l.h"
#include "controlgpio.h"
#include "pfijo.h"
#include "xcache_l.h"

// Bernstein
PFijo b[4][4];

// Reservamos espacio para los datos de entrada y de salida
PFijo entrada[4][4];
PFijo salida[4][4][4];

// Prototipos de funciones
PFijo nodoIntermedio(int i, int j, int r, int s);
XTime testSW(int n);
XTime testHW(int n);
void inicializaDatos();

int main(void) {
    // Activamos la cache
    XCache_EnableICache(0xc0000000);
    XCache_EnableDCache(0xc0000000);

    // Inicializamos los datos
    inicializaDatos();
    // Inicializamos los GPIOs
    InicializaGPIO();

    printf("--------------------\r\n");
    printf("-Comenzando tests-\r\n");
    printf("--------------------\r\n");

    for (int n = 10; n <= 1000000; n *= 10) {
        XTime horaSW = testSW(n);
        XTime horaHW = testHW(n);
```

```
        printf("%d iteraciones SW: %ld ciclos.\r\n", n, horaSW);
        printf("%d iteraciones HW: %ld ciclos.\r\n", n, horaHW);
    }

    printf("--------------------\r\n");
    printf("--Test terminado--\r\n");
    printf("--------------------\r\n");

    // Desactivamos cache y terminamos
    XCache_DisableDCache();
    XCache_DisableICache();
    return 0;
}

PFijo nodoIntermedio(int i, int j, int r, int s) {
    PFijo retorno = 0;
    PFijo B;
    for (int k = 0; k <= r; k++) {
        for (int l = 0; l <= s; l++) {
            B = prod(b[r][k], b[s][l]);
            retorno += prod(entrada[i+k][j+l], B);
        }
    }
    return retorno;
}

void inicializaDatos() {
    // Constantes de Bernstein
    b[0][0] = a_fijo(1.);
    b[0][1] = a_fijo(0.);
    b[0][2] = a_fijo(0.);
    b[0][3] = a_fijo(0.);
    b[1][0] = a_fijo(0.5);
    b[1][1] = a_fijo(0.5);
    b[1][2] = a_fijo(0.);
    b[1][3] = a_fijo(0.);
    b[2][0] = a_fijo(0.25);
    b[2][1] = a_fijo(0.5);
    b[2][2] = a_fijo(0.25);
    b[2][3] = a_fijo(0.);
    b[3][0] = a_fijo(0.125);
```

```
    b[3][1] = a_fijo(0.375);
    b[3][2] = a_fijo(0.375);
    b[3][3] = a_fijo(0.125);

    // Datos de entrada
    entrada[0][0] = a_fijo(0.1);
    entrada[0][1] = a_fijo(0.25);
    entrada[0][2] = a_fijo(0.66);
    entrada[0][3] = a_fijo(0.78);
    entrada[1][0] = a_fijo(0.1);
    entrada[1][1] = a_fijo(0.25);
    entrada[1][2] = a_fijo(0.66);
    entrada[1][3] = a_fijo(0.78);
    entrada[2][0] = a_fijo(0.1);
    entrada[2][1] = a_fijo(0.25);
    entrada[2][2] = a_fijo(0.66);
    entrada[2][3] = a_fijo(0.78);
    entrada[3][0] = a_fijo(0.1);
    entrada[3][1] = a_fijo(0.25);
    entrada[3][2] = a_fijo(0.66);
    entrada[3][3] = a_fijo(0.78);
}

XTime testSW(int n) {
    XTime horaSW = 0;
    // Limpiamos el timer
    XTime_SetTime(horaSW);

    // Computamos n veces la división por software
    for (int m = 0; m < n; m++) {
        for (int i = 0; i < 4; i++) {
            for (int j = 0; j < 4; j++) {
                salida[0][i][j] = nodoIntermedio(0,0,j,i);
                salida[1][i][j] = nodoIntermedio(0,i,j,3-i);
                salida[2][i][j] = nodoIntermedio(j,0,3-j,i);
                salida[3][i][j] = nodoIntermedio(j,i,3-j,3-i);
            }
        }
    }

    //Leemos la hora
```

```
    XTime_GetTime(&horaSW);
    return horaSW;
}

XTime testHW(int n) {
    XTime horaHW = 0;
    // Limpiamos el timer
    XTime_SetTime(horaHW);

    // Computamos n veces la división por hardware
    for (int m = 0; m < n; m++) {
        AjustaEscritura();
        for (int i = 0; i < 4; i++) {
            for (int j = 0; j < 4; j++) {
                Escribe(entrada[i][j]);
                Control(SET);
                Control(ESCRITO);
            }
        }
        Control(0x0);
        AjustaLectura();
        for (int i = 0; i < 4; i++) {
            for (int j = 0; j < 4; j++) {
                for (int k = 0; k < 4; k++) {
                    Control(GET);
                    salida[i][j][k] = Lee();
                    Control(LEIDO);
                }
            }
        }
        Control(0x0);
    }

    // Leemos la hora
    XTime_GetTime(&horaHW);
    return horaHW;
}
```

F.3.2. puntofijo.h

```
#ifndef PFIJO_H
```

```
#define PFIJO_H

typedef unsigned long long int Long_PFijo;
typedef unsigned int PFijo;

PFijo inline a_fijo(float n) {
    if (n == 1.f) {
        return 0xFFFFFFFF;
    } else {
        Long_PFijo a = 0x01;
        a <<= 32;
        return ((PFijo)(n * ((float)a)));
    }
}

PFijo inline prod(PFijo a, PFijo b) {
    Long_PFijo al = a;
    Long_PFijo bl = b;
    return (PFijo) ((bl * al) >> 32);
}

#endif
```

F.3.3. controlgpio.h

```
#ifndef CONTROLGPIO_H
#define CONTROLGPIO_H

#include "xgpio.h"
#include "xparameters.h"

#define MASK_CONTROL   0xFFFF0000
#define SET            0x00020000
#define ESCRITO        0x00100000
#define GET            0x00040000
#define LEIDO          0x00080000
#define OK             0x00000001
#define READY          0x00000002
#define INPUT          0xFFFFFFFF
#define OUTPUT         0x00000000
```

```
void InicializaGPIO();

#define Lee() Pgpio[0x00]
#define Escribe(x) Pgpio[0x00] = (x)
#define Control(x) Pgpio[0x08] = (x)
#define AjustaLectura() Pgpio[0x04] = INPUT
#define AjustaEscritura() Pgpio[0x04] = OUTPUT

extern volatile unsigned int* Pgpio;

#endif //CONTROLGPIO_H
```

F.3.4. controlgpio.cpp

```
#include "controlgpio.h"

volatile unsigned int* Pgpio;

//Inicializa los GPIOs
void InicializaGPIO() {
    XGpio gpio;

    //Inicializamos el GPIO.
    XGpio_Initialize(&gpio, XPAR_GPIO_DEVICE_ID);

    //Ajustamos la dirección del control
    XGpio_SetDataDirection(&gpio, 2, MASK_CONTROL);

    //Ponemos a 0 las señales de control
    XGpio_DiscreteWrite(&gpio, 2, 0x0);

    Pgpio = (volatile unsigned int*) gpio.BaseAddress;
}
```

Apéndice G

Maple: Un programa matemático

Maple es un programa matemático de propósito general capaz de realizar cálculos simbólicos, algebraicos y de álgebra computacional. Fue desarrollado originalmente en 1981 por el Grupo de Cálculo Simbólico en la Universidad de Waterloo en Waterloo, Ontario, Canadá. Su nombre proviene de MAthematical PLEasure (Placer Matemático).

Desde 1988 ha sido mejorado y vendido comercialmente por Waterloo Maple Inc. (también conocida como Maplesoft), una compañía canadiense con sede en Waterloo, Ontario. La última versión conocida a finales de 2006 es Maple 10.

Maple permite crear documentos técnicos ejecutables que ofrecen tanto la respuesta como el pensamiento detrás del análisis. Los documentos Maple combinan fácilmente cálculos simbólicos y numéricos, exploraciones, notaciones matemáticas, documentación, botones y sliders, gráficas y animaciones que pueden ser compartidas y reusadas por otras personas.

G.1. Introducción a Maple

G.1.1. Sintaxis

Algunas nociones básicas que hay que saber para escribir un programa en Maple:

- Salto de línea: cada vez que se pulsa `intro`, Maple ejecuta el conjunto de comandos que estamos escribiendo, pero se puede realizar un salto de línea sin que Maple realice el cálculo pulsando `Mayus + Return`.

- Fin de sentencias: Cada comando termina con punto y coma, "`;`", o dos puntos, "`:`". En el primer caso, Maple muestra el resultado de la instrucción, mientras que en segundo caso no se muestra nada por pantalla.
- Asignación: mediante el símbolo "`:=`".
- Carácter de repetición $: `x$3;` → x,x,x repite 3 veces la x.
- Ayuda: para obtener ayuda sobre algún término, se escribe el término precedido de una interrogación y terminado en punto y coma: `?array;`.
- Inclusión de paquetes: se puede cargar un paquete con la orden `with`. Ejemplo: `with(VectorCalculus);`
- Comentarios: se pueden insertar comentarios de dos formas:
  - Poniendo el carácter reservado `#` al principio del comentario. Todo lo que haya detrás en la misma línea se considera comentario.
  - Pulsando el botón de la barra de tareas que tiene una `T`. En este caso accedemos al modo texto y los comentarios se escriben en negro. Para volver al modo de comando se pulsa el botón [>.

G.1.2. Variables en maple

Toda palabra no reservada o letra es tomada como una variable simbólica. Si no tienen valores asignados, pueden utilizarse para realizar cálculo simbólico. Cuando lo tienen, sirven para realizar cálculo numérico.

- Cálculo numérico: `x := 3; y := 2; x*y;` → 6.
- Cálculo simbólico: `int(tan(x^2),x);`

$$\int \tan(x^2)dx$$

- Para hacer que una variable deje de tener valor (como en el primer ejemplo) y pueda utilizarse como en el segundo ejemplo, hay que desasignar su valor: `unassign('x');`

Los tipos se asignan de forma implícita, pero hay que ser coherente, ya que no se hace chequeo de tipos. Algunas peculiaridades de los valores asignados a las variables:

- Coma decimal: Se representa mediante el punto.
- Concatenación: el símbolo "`||`" sirve para concatenar dos cadenas. Ejemplo:
  - `cadena := ‘‘hola’’ || ‘‘adios’’;`
  - `cadena := ‘‘hola adios’’;`
- Rangos: para representar un rango se puede utilizar `..`, por ejemplo:`2..4`. Esto es útil especialmente a la hora de dibujar funciones, explicado más adelante.

G.1.3. Tipos de datos

En Maple no es necesario declarar explícitamente los tipos. No obstante, cuando los tipos se declaran de forma explícita, se realiza chequeo de tipos. Ejemplo:

```
f := proc(x :: integer) :: integer
   return(x!);
end;
```

- Comprobación de tipo: se puede comprobar si un objeto tiene un determinado tipo con la función `type(`*`objeto`*`,`*`tipo`*`)`, que devuelve `true` cuando el tipo es el indicado, y `false` en caso contrario.
- Conversión de tipos: con la orden `convert(`*`objeto`*`,`*`tipo`*`)` se realiza una conversión de tipos siempre que sea posible; es decir, que si por ejemplo se intenta convertir 3,0 a entero, lo hará, pero 2,5 no, porque habría que redondear o truncar primero.

Tipos estructurados

1. Secuencia: lista de valores separados por comas: `secu := 1,2,3,4;`. Para acceder a un elemento de la secuencia `secu[i];` Generación de una secuencia: con la orden `seq`, fijando una operación y un rango, se genera una lista. Ejemplo: `seq(x^2,n=1..10);`
2. Lista: array con el índice inicial en 1. Pueden anidarse (listas de listas de ...). Utilizaremos principalmente las listas como parámetros de entrada y salida de los procedimientos. Las listas y las secuencias se pueden convertir de forma sencilla unas en otras:

- Declarar una lista vacía: `lista:=[]`
- Pasar de secuencia a lista: `lista := [secu];`
- Pasar de lista a secuencia: `secu := lista[];`
- Concatenar dos listas: `dobleLista := [lista[],lista[]];`
- Número de operandos de una lista: `nops(lista);`
- Añadir elementos a una lista: `lista:=[op(lista),elemento]`
- Acceder a un elemento de la lista: `lista[i];`

3. Array: similar a las listas, salvo que comienzan con el índice 0. Ejemplo: `matriz := array(0..2,1..2);`

   Si deseamos inicializar el array al crearlo, podemos hacer: `matriz := array(0..2,1..2,[[1,2],[3,4],[5,6]]);`

   También se puede inicializar un array con una lista: `matriz := array(0..2,lista);`

   O la operación contraria, convertir un array a lista: `convert(`*`matriz`*`,list);`

   Existe también un tipo específico matriz, que se define del siguiente modo: `m:=matrix(1..3,1..3)`, que es equivalente a un array de dos dimensiones. Para usarlo hay que incluir la biblioteca `with(LinearAlgebra);`

4. Conjunto: se define con llaves. Ejemplo: `conj := {A,B,C,A};`

   La diferencia con las listas es que en los conjuntos no importa el orden, mientras que en las listas si. De esta forma las listas `[1,1,2,2]` y `[1,2,1,2]` son diferentes, mientras que los conjuntos `1,1,2,2` y `1,1,2,2` son iguales.

5. Vector: es el tipo de datos vector matemático. Ejemplo: `v := Vector([X,Y,Z]);`

G.2. Estructuras de control

En Maple, como en cualquier lenguaje de alto nivel, se dispone de varias estructuras para controlar el flujo de programa.

- IF:

  1. `if` *`condición`* `then` *`instrucciones separadas por ;`* `end if;`
  2. `if` *`condición`* `then` *`instrucciones separadas por ;`* `else` *`más instrucciones`*`; end if;`
  3. Se puede sustituir `end if;` por `fi;`

4. O de otra forma alternativa: `‘if‘(`*condición*`,valor en caso positivo,valor en el otro caso`*);*
   Ejemplo: `‘if‘(x=3,2,1);`

- FOR:

1. `for` *variable* `from` *inicio* `to` *final* `by` *paso* `do` *instrucciones*`; end do;`
2. Se puede sustituir `end do;` por `od;`
3. La parte del `by` *paso* se puede omitir (el paso será 1)

- WHILE:

1. `while (`*condición*`) do` *instrucciones*`; end do;`
2. En este caso también se puede cambiar el `end do;` por `od;`

También se pueden definir funciones, o procedimientos de una sola línea. La forma general es la siguiente:

```
f := (parámetros entrada) -> [parámetros salida];
```

Por ejemplo: `f := x ->x^2 * sin(x)`. Después de definirla, se puede llamar a la función simplemente con `f(Pi);`, por ejemplo.

Si se desea poner más de un parámetro de entrada, se colocan entre paréntesis y separados por comas. Si se desea que tengan más de un parámetro de salida, se colocan entre corchetes y separados por comas también:

```
h := (x,y) -> [x*y, x/y];
```

También se pueden incluir dentro de las funciones estructuras de control simplificadas: `escalón := x ->‘if‘(x >= 0, 1, 0);`

G.3. Procedimientos

La forma general de un procedimiento es la siguiente:

```
procedimiento := proc (n :: integer)::integer;
#Declaraciones locales
   local aux;
#Cuerpo
   aux := 1;
   if n=0 then return aux else aux := n*procedimiento(n-1);
```

```
   fi;
return aux;
end proc;
```

Una vez definido, podríamos llamar a este procedimiento simplemente con `procedimiento(5);`.

Si quisiéramos declarar variables globales a toda la hoja dentro de un procedimiento, basta con escribir `global` *`variable`*, de forma análoga a como hemos declarado las variables auxiliares en el procedimiento auxiliar.

No es necesario declarar el tipo de las variables de entrada y el valor de salida del procedimiento de forma explícita, entonces maple asignaría los tipos de forma automática y no haría chequeo de tipos.

G.3.1. Procedimientos con argumentos de entrada variables

En ocasiones será interesante tener un procedimiento que tenga un número aleatorio de parámetros de entrada. Por ejemplo, el siguiente procedimiento muestra el número de parámetros de entrada, y luego muestra todos ellos:

```
prueba := proc();
   print(nargs);
   print(seq(args[i],i=1..nargs);
end proc;
```

G.4. Algunas funciones predefinidas

- Raíz cuadrada: `sqrt();`
- Función subs: la función subs realiza una sustitución de un valor por otro dentro de cualquier estructura o función.

  ```
  lista:=[amarillo, verde, rojo, azul, negro, blanco];
  subs(negro=naranja,lista);
  ```

  El resultado sería la lista [amarillo, verde, rojo, azul, naranja, blanco].

  ```
  e:=2*x^2;
  subs(x=2,e);
  ```

  Devuelve el resultado de la expresión sustituyendo x por 2.

- Convertir expresiones en funciones: existen dos posibles comandos para realizar esta tarea.
  - Comando Unnaply:

```
e:=a*x^2+b*exp(y);
f:=unnaply(e,x,y);  #especificamos las variables de la función
f(0,1);
```

  - Función subs: lo siguiente es equivalente a lo anterior.

```
e:=a*x^2+b*exp(y);
h:=subs(body=e,(x,y->body));
h(0,1);
```

- `eval()`: devuelve el valor de la expresión o variable entre paréntesis. Se puede ser un poco más específico e indicar el tipo que se tiene que devolver con `evalf()` para reales o `evalb()` para booleanos, etc.
- Funciones trigonométricas: `sin()`, `cos()` o `tan()` .
- Función exponencial: `exp()`.
- Límite de una función: `limit(cos(x)/x,x=infinity);`.

  También se puede calcular el límite por la derecha o por la izquierda de un valor: `limit(cos(x)/x,x=Pi/2,left)` o `limit(cos(x)/x,x=Pi/2,right)`.

  Se puede usar la función `Limit()` de la misma forma, pero que solo sirve para imprimir la fórmula por pantalla.
- Derivadas: existen diferentes posibilidades:
  - Derivada de una expresión respecto a una variable: `diff(x^2,x);`
  - Derivada de una función:

```
  f:=x->exp(-2*x);
diff(f(x),x);
  diff(f(x),x,x);  #derivada respecto de x dos veces
```

  - Derivada de una función de dos variables:

```
  g:=(x,y)->x^2*y + y^2*x^3;
diff(g(x,y),x);
  diff(g(x,y),x,y);  #derivada respecto a x y luego respecto a y
```

  - Función Diff: no realiza la derivada, si no que solo sirve para imprimir la derivada por pantalla de forma elegante.

- Integrales:

  - Integrales indefinidas: `int(sin(y)*cos(y),y)`, devuelve la integral respecto a la variable y, si encuentra respuesta.
  - Integrales definidas: es suficiente con especificar el intervalo de integración: `int(1/x,x=2..4)` $\rightarrow \ln(4) - \ln(2)$.
  - Integral respecto a varias variables:`int(int(sin(x)*cos(y),x),y)`.
  - Impresión por pantalla: `Int(sin(y)*cos(y),y)`, devuelve los signos de integración.

G.5. Representación gráfica

La forma más sencilla de representar una función en Maple es la siguiente:

- Se asigna la función a representar a una variable
- Se dibuja la variable en la pantalla
- Ejemplo:

```
b := plot(sin);
plots[display](b);
```

También se puede representar utilizando directamente una función e indicando el rango:

```
plot(escalón(x), x=-2..2);
```

Antes que nada, para usar la función plot y todos sus derivados hay que incluir la librería plots: `with(plots)`. A partir de aquí, ya podemos trabajar con el amplio rango de funciones que nos ofrece dicha librería.

G.5.1. Representación 2D

La forma general de la función `plot` para representar funciones en 2D es la siguiente:

```
plot(función, rango, opción 1, ...,  opción n);
```

Por ejemplo:

```
plot(sin(10*x)+sin(11*x),x=0..3,thickness=2,numpoints=50);
```

También puede utilizarse la función plot para dibujar una lista de objetos. Por ejemplo, si queremos dibujar una lista de puntos enlazados basta con escribir:

```
plot([[2,3],[2,4],[3,5]]);
```

Pero, dentro de la biblioteca plots existen un grupo de funciones mucho más específicas que la función `plot`, que permiten ampliar las opciones de lo que queremos mostrar por pantalla.

Por ejemplo, para dibujar una lista de puntos no enlazados:

```
pointplot([[2,3],[2,4],[3,5]], opciones);
```

Las opciones se separan con comas y las más destacadas son:

- `color`:asigna un color a los puntos, se asigna especificando directamente el color (blue, green, etc) o definiendo uno nuevo (COLOR(RGB,0.5,1,0.5)).
- `style = line`:conecta los puntos entre sí, definiendo el tipo de linea que los une.
- `thickness = num`: define el grosor de la linea definida en el apartado anterior.
- `symbol = BOX`: dibuja cubos en lugar de puntos. Se puede ampliar con gran cantidad de símbolos como `diamond`,`cross`,`circle`, etc.
- `symbolsize = num`: define el tamaño de los símbolos definidos en el apartado anterior.

Otra función que puede ser interesante es la que nos permite dibujar polígonos dada la lista de vértices. Por ejemplo, un polígono relleno de azul:

```
polygonPlot([[2,5],[3,1],[4,6],[5,3]],color=blue)
```

G.5.2. Representación 3D

Es equivalente a la representación 2D pero especificada para dos variables dentro del espacio. Algunos ejemplos:

```
plot3d(sin(x)*cos(y),x=-Pi..Pi,y=-Pi..Pi);

plot3d(sin(10*x)*cos(y^2),x=-Pi..Pi, y=-Pi..Pi,
                      orientation=[-45,12],grid=[100,100]);

plot3d([cos(s)*cos(t),cos(s)*sin(t),sin(s)],
                      s = -Pi/2..Pi/2, t=0..2*Pi);
```

G.5.3. Otras funciones de utilidad

Existe una función que nos permite visualizar una lista, conjunto o array de estructuras plot. La sintaxis es la siguiente:

```
display( lista, opción 1, ..., opción n);
```

Por ejemplo, para funciones 3d, dibujamos una esfera y una curva en la misma imagen:

```
esfera := plot3d([cos(s)*cos(t),cos(s)*sin(t),sin(s)],
     s = -Pi/2..Pi/2, t=0..2*Pi):
curva := plot3d([cos(s)*cos(s),cos(s)*sin(s^2),sin(s),
     s=-Pi/2..Pi/2],thickness=10,color=black):
display3d([esfera,curva],axes = boxed);
```

La opción "axes", muestra los ejes de coordenadas en la imagen.

G.5.4. Animación

Maple ofrece la posibilidad de hacer animaciones. En éstas, se representa una función que varía en el tiempo. Las funciones requieren pues, un parámetro adicional a los suyos propios. Así por ejemplo en una función 2D, hay que especificar el rango de la variable en el espacio y el rango de la variable temporal. Por ejemplo:

```
animate(sin(t*x),x=-Pi..Pi,t=1..5);
```

Con esta instrucción nos sale una imagen normal, pero si pinchamos sobre ella, aparecen unos botones en la barra de herramientas similares a los de un vídeo, que sirven para controlar la animación (stop, play, rewind, fast forward).

Se pueden modificar algunas opciones, por ejemplo, el número de fotogramas de la animación:

```
animate(sin(t*x),x=0..2*Pi,t=1..5,frames=12);
```

El caso tridimensional es análogo al anterior, aunque variando ligeramente la sintaxis.

```
animate3d(cos(t*x)+sin(t*y),x=-Pi..Pi,x=-Pi..Pi,t=1..5);
```

Apéndice H

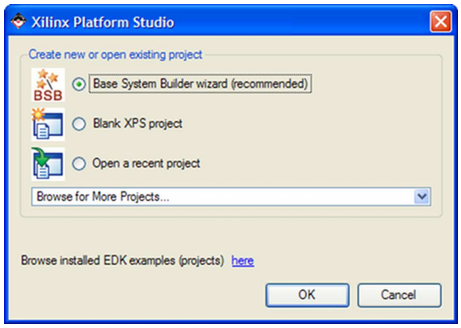
Xilinx EDK: *Embedded Development Kit*

Para realizar la integración del hardware con el software implementado, tuvimos que utilizar la herramienta Xilinx EDK, un entorno de desarrollo integrado hardware–software. Éste permite utilizar una variedad de placas de desarrollo con gran cantidad de hardware en forma de *soft–cores*, es decir, hardware del que se proporciona una *netlist* y se implementa directamente en la propia FPGA de la placa.

Aunque se trata de un entorno bastante simplificado, en un principio resulta bastante confuso utilizarlo, por lo que hemos decidido crear un pequeño manual centrado en cómo lo utilizamos nosotros.

H.1. Creación de un proyecto en blanco

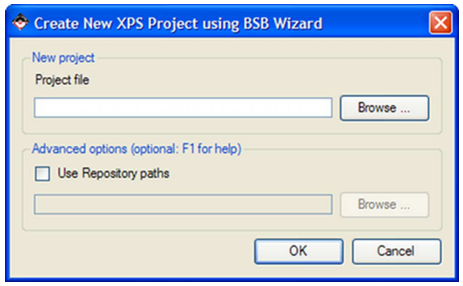
Ejecutamos el *Xilinx Platform Studio* y aparece un pequeño diálogo que nos permite crear un proyecto básico:



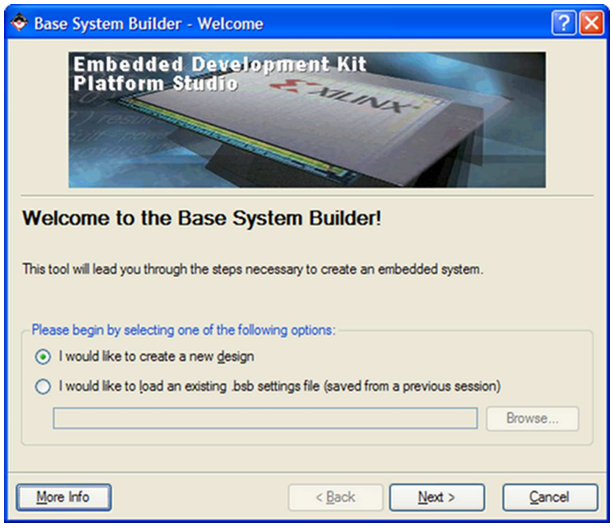
Seleccionamos la opción *Base System Builder wizard*, que más adelante nos permitirá configurar el hardware que deseamos utilizar, y pulsamos sobre

el botón *Ok*.

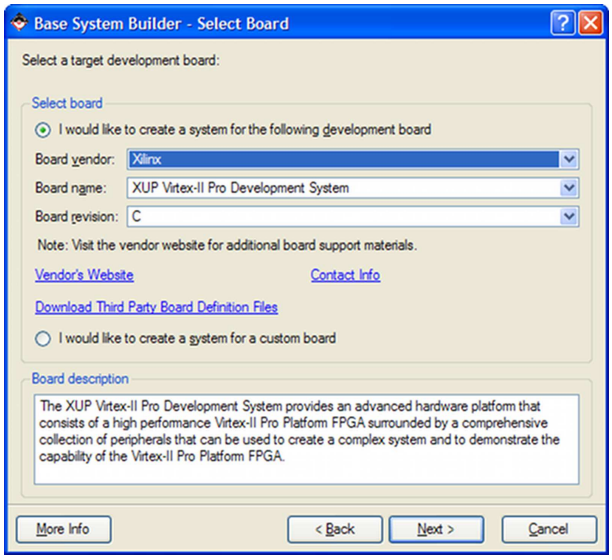
Tras realizar el paso anterior, aparece el diálogo siguiente:



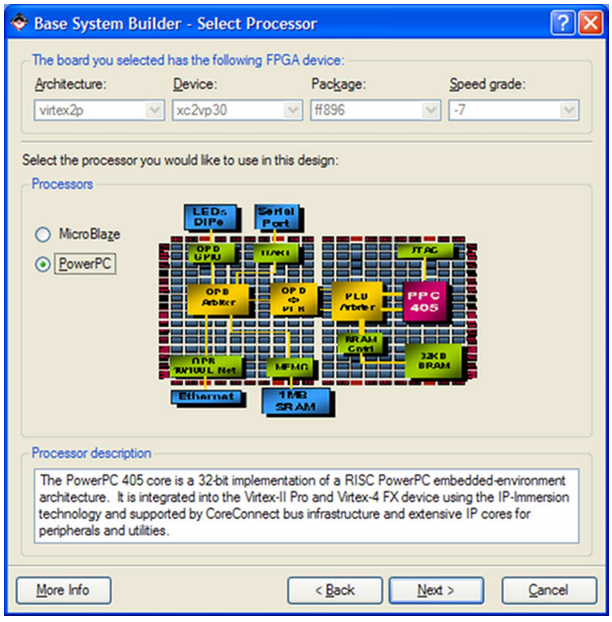
A continuación, en el grupo *New Project*, debemos elegir la ruta donde guardaremos el proyecto. Es muy importante elegir una ruta que no esté dentro del propio directorio del EDK —por defecto, `C:\EDK`— y, además, que no contenga espacios en el nombre. Si no fuese así, es muy probable que se produzcan errores durante el diseño cuya relación con esto no es obvia, y sería difícil dar con la solución. Adicionalmente podemos seleccionar alguna carpeta donde existan bibliotecas de diseños hardware que vayamos a usar para nuestro proyecto (como, por ejemplo, las de la placa que vayamos a usar). Pulsamos el botón *Ok* y aparece el siguiente diálogo:



Para nuestro propósito, elegiremos la casilla *I would like to create a new design*, y pulsamos el botón *Next*. Aparece la imagen siguiente:



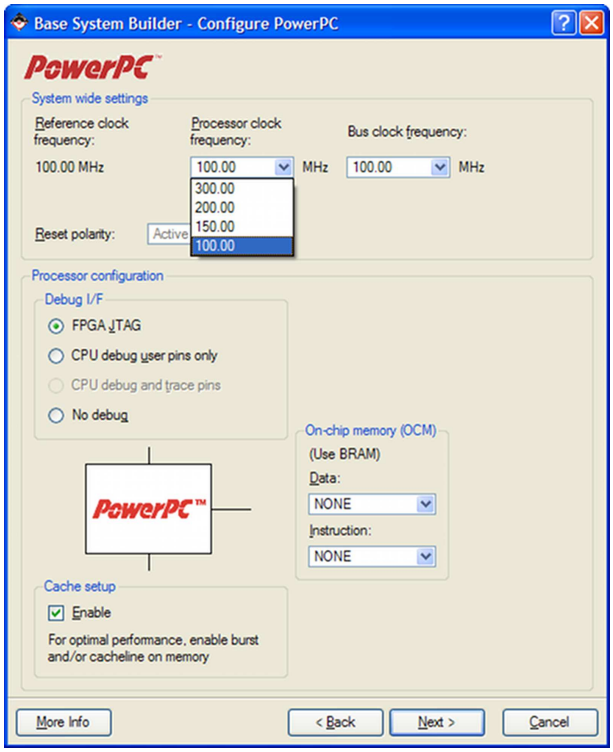
Elegimos el tipo de placa que vamos a utilizar en nuestro desarrollo y pulsamos *Next*, con lo que aparece lo siguiente:



Ahora es el momento de seleccionar el tipo de procesador que vamos a utilizar en nuestro diseño.

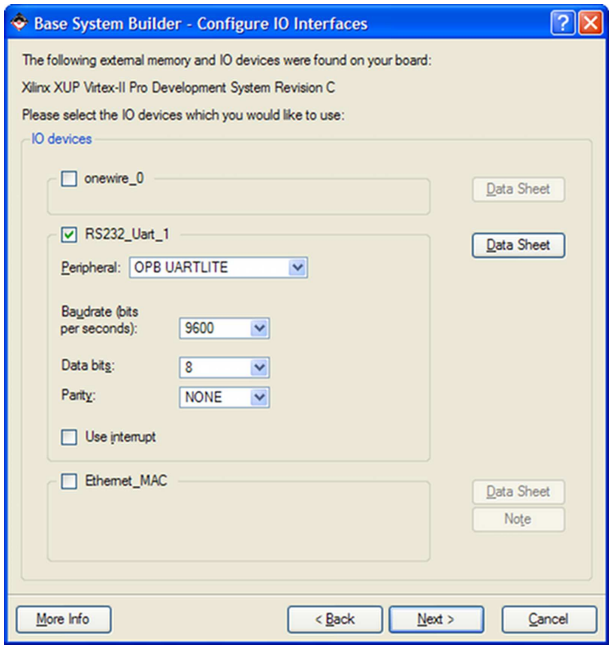
En el caso de elegir el procesador PowerPC, utilizaremos uno de los dos procesadores *PowerPC 405* físicamente presentes en la Virtex II Pro —de los cuales, uno se utiliza directamente por el programador y el otro se puede utilizar para depuración—. Por otro lado, si seleccionamos *MicroBlaze*,

utilizaremos la FPGA para implementar un procesador en VHDL diseñado por Xilinx. En nuestro caso, seleccionaremos el *PowerPC* y pulsaremos sobre *Next*.

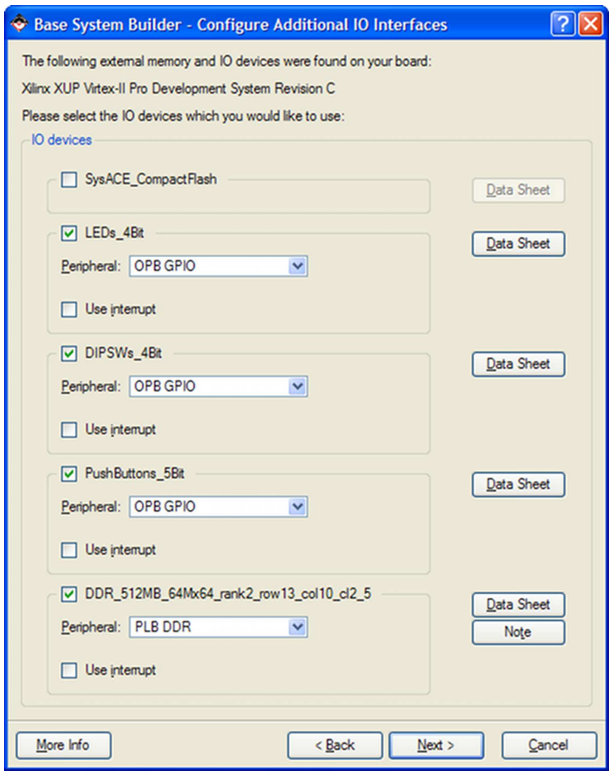


En este diálogo elegiremos la frecuencia de reloj del procesador y del bus; dejaremos ambas a 100MHz. En cuanto a las opciones de depuración, elegiremos *FPGA JTAG*. Habilitaremos la cache marcando la casilla correspondiente, y no utilizaremos la memoria interna del chip, ya que nuestro programa no cabe en ella. En el caso de utilizar dicha memoria, la utilización de cache resulta innecesaria, pues ambas memorias están al mismo nivel en la jerarquía de memoria.

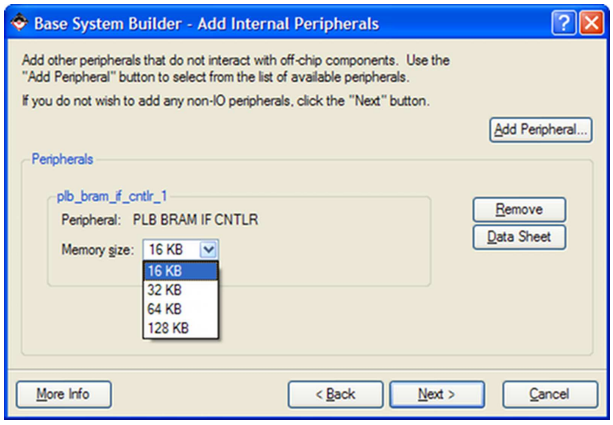
Es importante elegir algún tipo de depuración si no elegimos memoria interna. Si nuestro programa se carga en memoria externa, tendremos después 2 opciones para cargarlo: o bien programamos algún tipo de cargador que se ejecute desde una memoria flash, o bien cargamos el programa en memoria utilizando un depurador. Si elegimos la segunda opción, tendremos que elegir las opciones de depuración. Pulsamos el botón *Next*, con lo que procedemos a elegir los distintos componentes hardware de nuestro sistema.



Para nuestro proyecto, como hardware de entrada–salida, utilizaremos sólo el puerto serie, por lo que seleccionaremos el componente `RS232_UART`. Pulsamos sobre el botón *Next*.

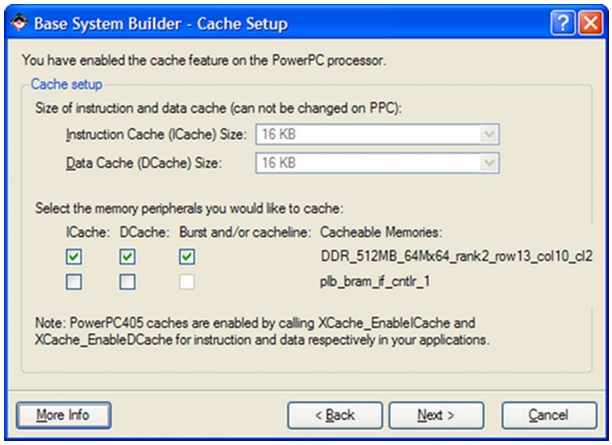


En este diálogo, seleccionaremos algunos periféricos que utilizaremos. En concreto, utilizaremos los botones, los *switches* y los *leds*, así como la memoria *DDR_512MB* señalada. Es importante saber que, si en nuestro proyecto vamos a utilizar GPIOs, deberemos seleccionar en este fase alguno de los periféricos que hemos dicho —botones, leds o interruptores— porque, de lo contrario, no se crean ciertos archivos que luego nos darán problemas a la hora de implementar la parte software del proyecto. Una vez elegidos estos componentes, pulsamos *Next*, con lo que aparece la siguiente imagen:

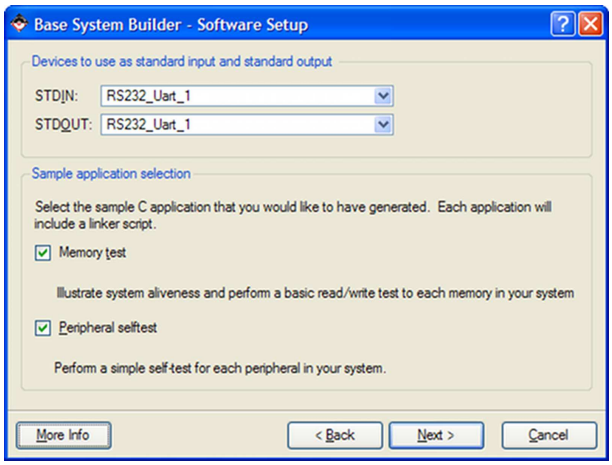


Ahora deberemos seleccionar otro tipo de memoria interna del chip. Ya que anteriormente no hemos seleccionado la otra memoria, utilizaremos ésta. Se trata de un tipo de memoria que se conecta con el procesador por medio del bus PLB[1], y que nos servirá para almacenar el *bootloop*. Ya que nuestro programa se almacenará en memoria dinámica, cuando la placa se encienda, el procesador deberá tener algún programa que ejecutar. El *bootloop* es un programa que sólo consta de una instrucción: salto incondicional a esa misma instrucción. De este modo, cuando la placa se conecte, ejecutará el *bootloop*. Si no se tuviera ese programa, ya que la memoria dinámica contiene inicialmente valores aleatorios, se pondría a ejecutar código aleatorio. Aunque podría no ocurrir nada, lo normal es que al ejecutar basura se genere una excepción o, en cualquier caso, que el procesador vaya un estado desconocido. Podría llegar a ocurrir, aunque la posibilidad es muy remota, que se modificase el valor de algún registro que controle buffers tri-estado y que, entonces, se produjese alguna avería eléctrica en la placa. La posibilidad es remota, pero aún así es preferible utilizar el *bootloop*. Pulsamos sobre el botón *Next* y aparece el siguiente cuadro de diálogo:

[1] *Processor Local Bus*, un tipo de bus de alta velocidad que se utiliza para conectar periféricos que requieran un alto ancho de banda, como la DDR.

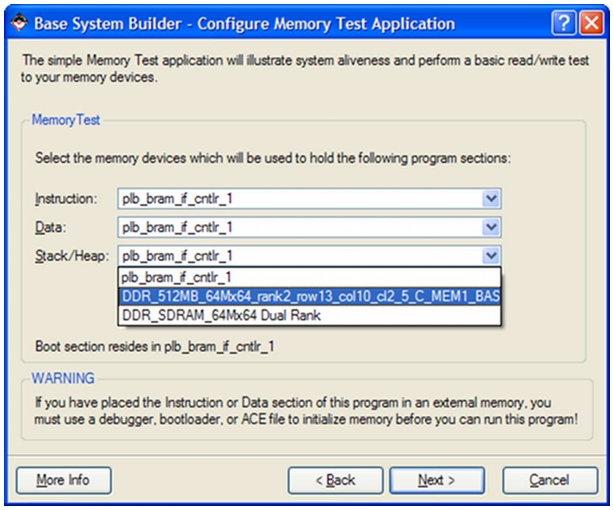


En este diálogo elegiremos qué memorias utilizarán caches de datos o instrucciones. Seleccionamos la DDR, pero no es necesario seleccionar la otra memoria porque sólo contendrá el *bootloop*. Después, pulsamos *Next* y llegamos hasta el siguiente diálogo:

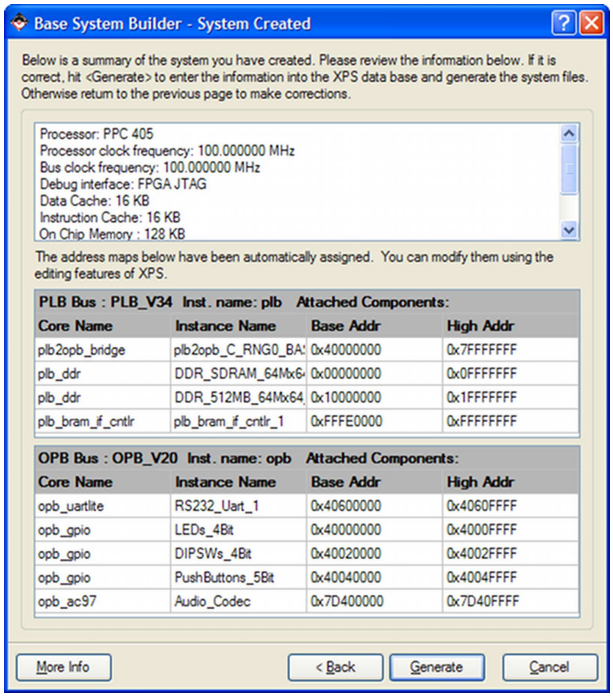


Debemos seleccionar ahora los dispositivos de entrada–salida que utilizaremos como entrada y salida estándar en la parte software. Con entrada y salida estándar nos referimos al sentido de *stdin* y *stdout* de UNIX, ya que las herramientas que se utilizan para compilar y enlazar los programas que se ejecutan en la placa son del proyecto GNU. En nuestro caso, utilizaremos el *RS232* tanto para entrada como para salida.

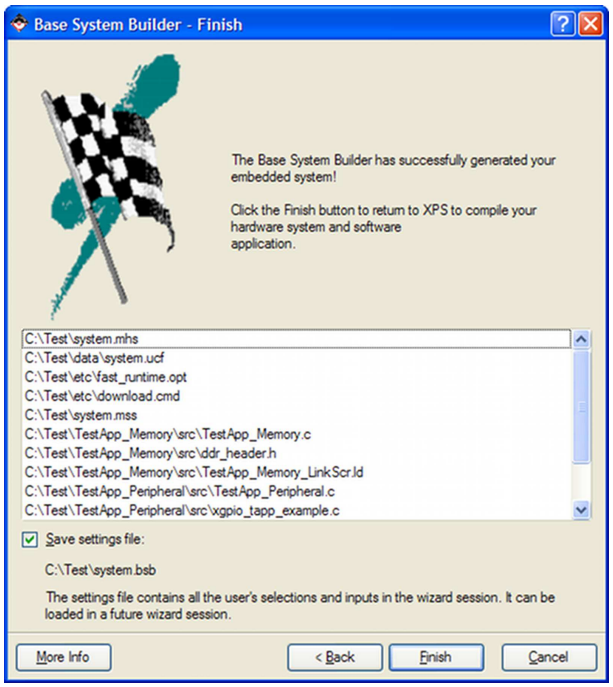
Después, elegiremos qué programas de muestra generará el asistente. Marcaremos las casillas correspondientes a los *test* de memoria y de periféricos y pulsaremos *Next*.



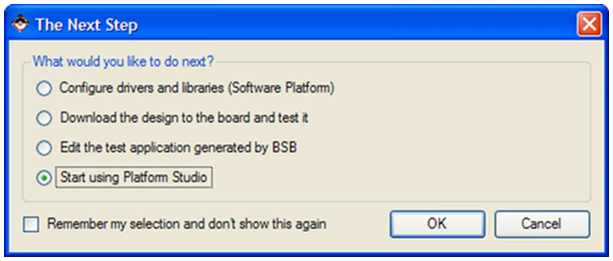
Ahora debemos seleccionar en qué memoria almacenaremos las instrucciones, los datos y el montículo del programa de verificación de memoria. Para ver cómo se hace en el caso de los programas en memoria estática, colocaremos datos e instrucciones en la memoria estática, mientras que para el montículo escogeremos la memoria dinámica. Pulsamos *Next*, y hacemos igual para la otra aplicación. Pulsamos una vez más *Next* y aparece el diálogo siguiente:



Aparece un sumario con el hardware que hemos configurado en nuestro proyecto. Comprobamos que todo es correcto y pulsamos el botón *Next*.



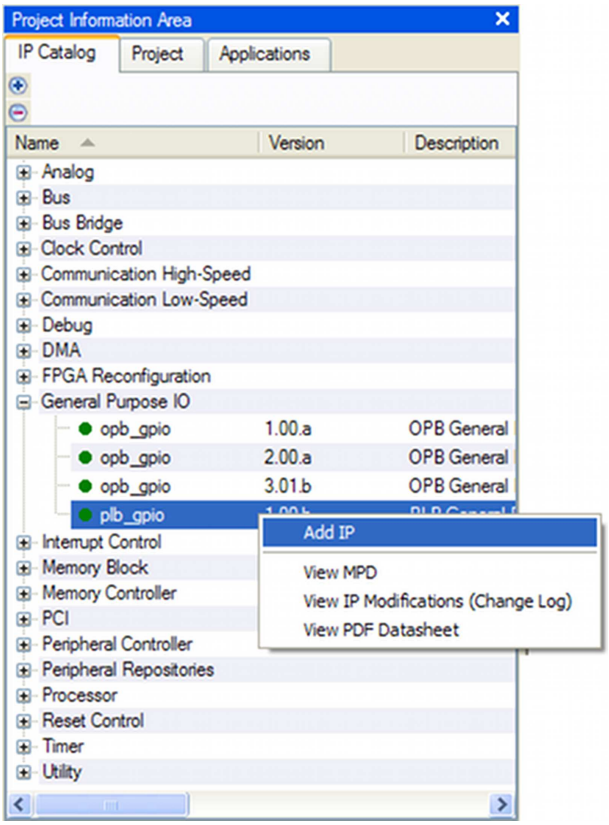
Para comenzar a trabajar en nuestro proyecto, sólo falta pulsar una vez más en el botón *Next*, con lo que aparece el siguiente diálogo:



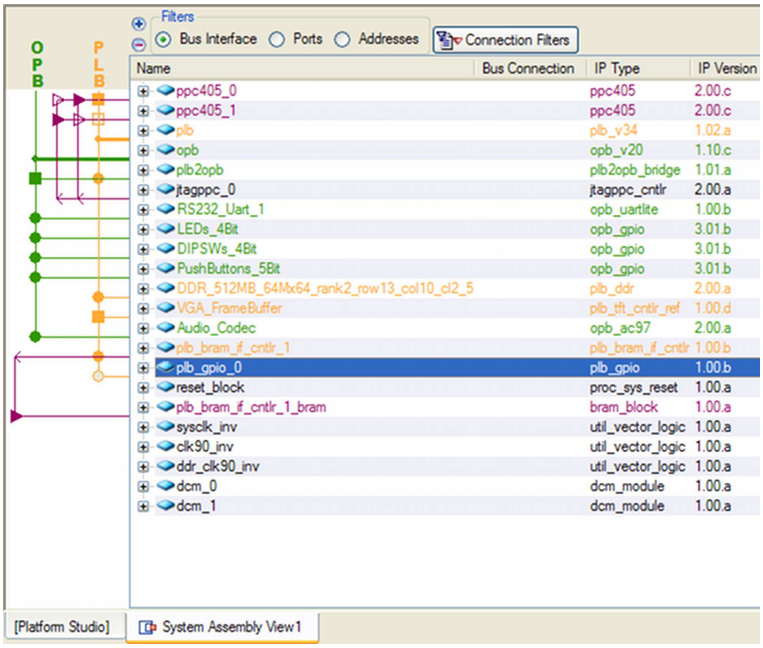
H.2. Selección de componentes hardware

Elegimos la opción *Start using Platform Studio* y pulsamos el botón *Ok*, con lo que se cierra el diálogo. Aparece entonces la pantalla principal de *Xilinx Platform Studio*. Como nuestro proyecto utilizará un hardware que se conecta con el procesador por medio de GPIOs, añadiremos un periférico que implemente los GPIOs y que se conecte con el procesador por medio del bus PLB. Para ello, en la parte izquierda de la ventana principal, en el *Project*

Information Area, y seleccionamos la pestaña *IP Catalog*. En ella, aparecen todos los periféricos que se pueden añadir al sistema ordenados por categorías; se trata del conjunto de entidades VHDL que están en los repositorios del proyecto. Hay que tener en cuenta que no todos los periféricos se pueden conectar a los dos tipos de procesador (*PowerPC* y *MicroBlaze*), el entorno no advierte al tratar de añadir un periférico diseñado para un procesador en otro, y aparecerán problemas más adelante.



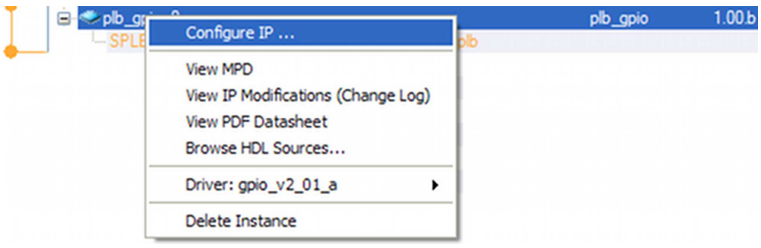
Desplegamos la categoría *General Purpose IO* y pulsamos, con el botón derecho del ratón, sobre el periférico `plb_gpio`. En el menú desplegable que aparece, pulsamos sobre la opción *Add IP*, con lo que el periférico se añadirá al sistema.



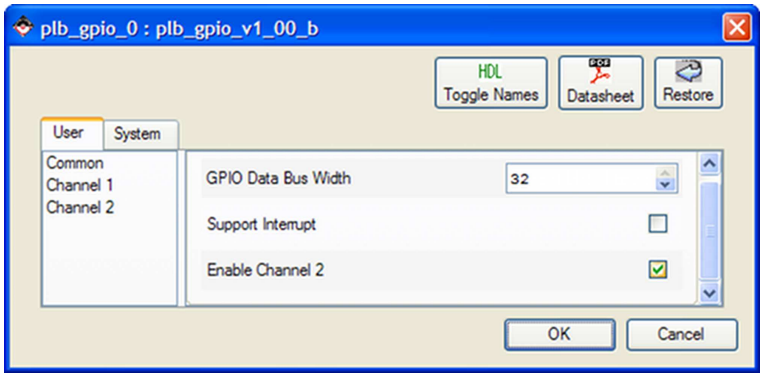
Como podemos ver en el centro de la ventana, si desplegamos la pestaña *System Assembly View* podemos ver que está el componente que hemos elegido. Sin embargo, como vemos en el círculo hueco que tiene en la conexión con el bus PLB, aún no está conectado al sistema. Para conectarlo, deberemos desplegar el dispositivo como en la siguiente figura:

Una vez desplegado, vemos que en la columna de conexión pone *No connection*. Para conectar este periférico al bus PLB deberemos pulsar sobre *No connection*, con lo que aparecerá un menú desplegable en el que seleccionaremos la opción *PLB*.

Una vez conectado el dispositivo, es necesario configurarlo: número de canales, dirección de los datos, valores de reset, espacio de direcciones, anchura de datos, etc. Para ello, como vemos en la figura siguiente, pulsamos con el botón derecho de ratón sobre el dispositivo, y en el menú desplegable que aparece, seleccionamos la opción *Configure IP*:



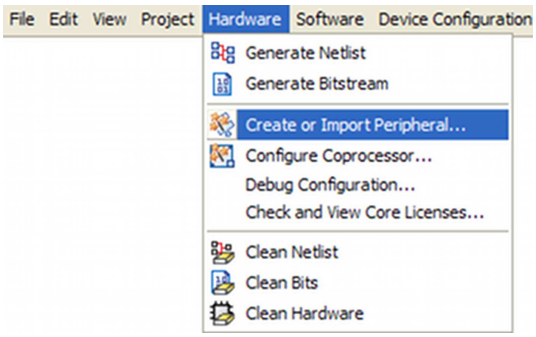
Tras realizar la acción anterior, aparecerá el siguiente diálogo:



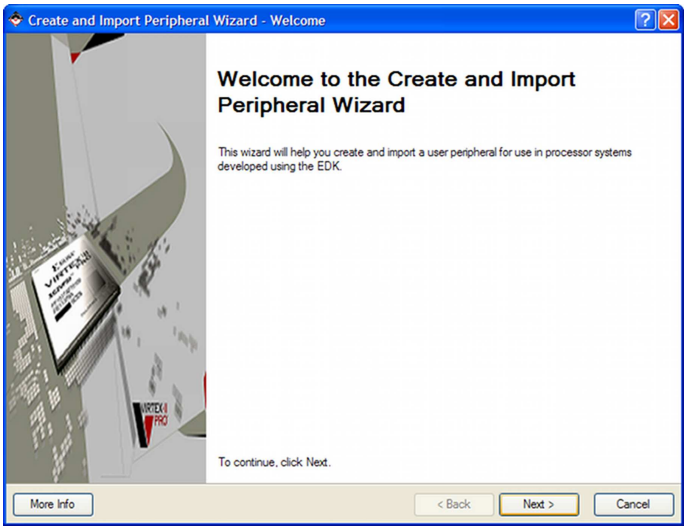
Primero modificaremos los parámetros que se muestran en la pestaña *User*. La anchura de datos por defecto del periférico es de 32 bits, por lo que no hay que modificarla. Sin embargo, por defecto no se activan los 2 canales del GPIO, por lo que es necesario marcar la casilla *Enable Channel 2*, que nos permitirá tener en total 64 señales. La casilla relativa a las interrupciones se deja sin marcar, pues no se utilizará. Pulsamos ahora sobre la pestaña *System*, con lo que aparece el siguiente panel:

Ahora, modificaremos los valores de dirección inicial y dirección final. Con el fin de tomar una zona del espacio de direcciones tal que no se solape con el del resto de dispositivos, tomaremos los valores `0x80080000` y `0x8008ffff` como direcciones base y final. Pulsamos el botón *Ok* para cerrar el diálogo.

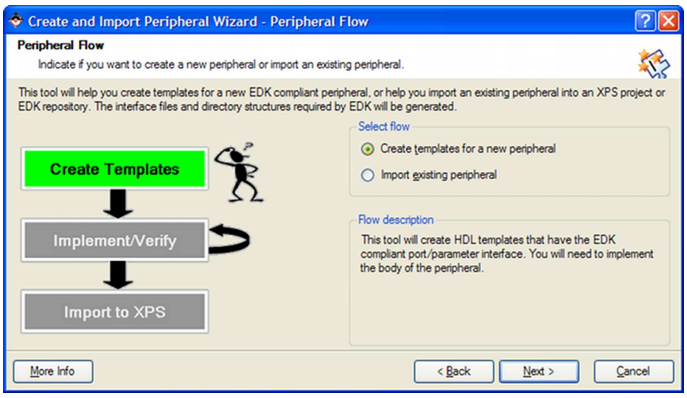
Tras importar los periféricos que proporciona *Xilinx*, crearemos una plantilla del hardware que vamos a integrar en el sistema. Para ello, pulsamos sobre la acción *Create or import peripheral* del menú *Hardware*, como se ve en la siguiente figura:



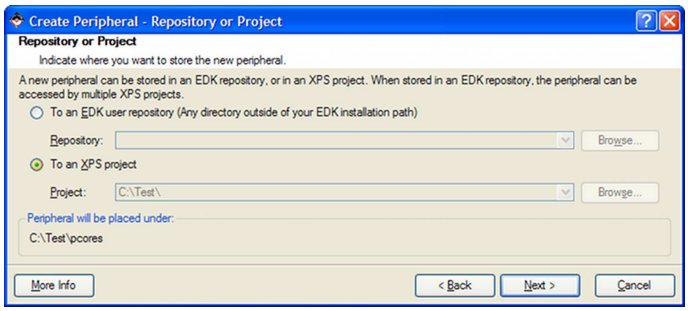
Aparece un asistente que nos guiará en la creación de un nuevo hardware:



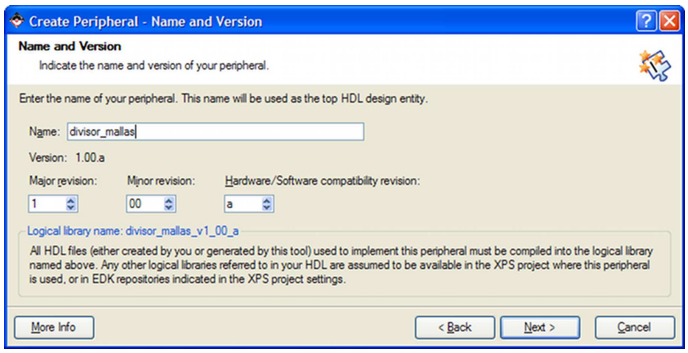
Pulsamos el botón *Next*, y aparece el siguiente diálogo:



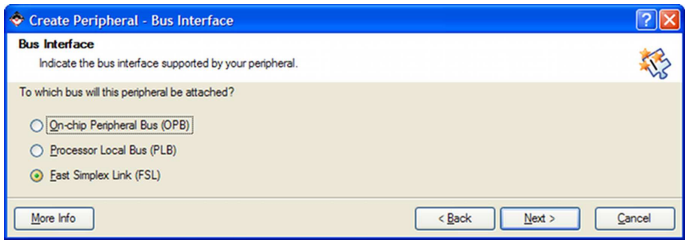
De las opciones que proporciona, elegimos *Create templates for a new peripheral*. El asistente nos dará a elegir entre algunas opciones y creará los archivos básicos para crear el periférico, tanto de configuración como de VHDL. Pulsamos el botón *Next*, que nos lleva hasta la siguiente imagen:



Este diálogo nos permite escoger dónde se almacenará el periférico. Podemos escoger entre un repositorio externo, de forma que el periférico se pueda compartir con otros proyecto, o un repositorio local al proyecto. En nuestro caso lo haremos local, por lo que seleccionamos la opción *To an XPS project* y pulsamos el botón *Next*. El diálogo ahora muestra lo siguiente:

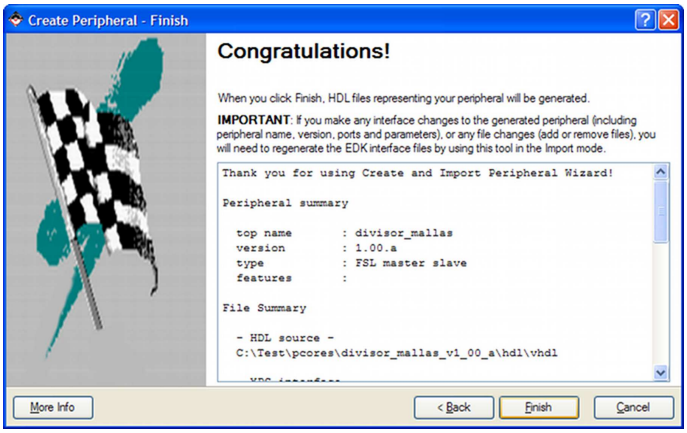


Debemos elegir ahora el nombre[2] del periférico. Introducimos, además, el número de versión, que nos será útil si en el futuro cambiamos la implementación de este hardware. Pulsamos sobre *Next* para seguir el proceso.



[2]No debe contener espacios.

Elegimos ahora el tipo de bus al que se conectará el procesador. En nuestro caso no queremos conectarlo a ningún bus estándar, sino que estableceremos nuestras propias conexiones. Para ello, elegimos una opción cualquiera y, más adelante, implementaremos nuestro propio tipo de conexión. Podemos ver en la figura que hemos seleccionado una conexión con *Fast Simple Link*, un tipo de conexión sólo disponible en procesadores *MicroBlaze* (no en los *PowerPC*). Pulsamos en siguiente y nos lleva a la siguiente imagen:



Para terminar la creación, sólo hay que pulsar el botón *Finish*.

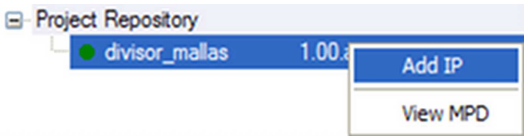
Una vez hayamos creado la plantilla para nuestro dispositivo, llega el momento de crearlo. Para ello, vamos hasta la carpeta *pcores* dentro de nuestro proyecto, donde se almacenan los dispositivos que hemos creado. Modificamos sus archivos *.vhd* y, después, modificamos los archivos del dispositivo que sirven para indicar al *XPS* el tipo de conexión que realizará nuestro periférico. Para ello, buscamos un archivo `.mpd`, que contiene una descripción de las señales de entrada y salida de nuestro dispositivo. Sustituimos el contenido de dicho archivo por lo siguiente:

```
BEGIN divisor_mallas

## Peripheral Options
OPTION IPTYPE = PERIPHERAL
OPTION IMP_NETLIST = TRUE
OPTION CORE_STATE = DEVELOPMENT
OPTION HDL = VHDL

## Peripheral ports
PORT clk = "", DIR=I, SIGIS=CLK
PORT rst = "", DIR=I, SIGIS=RST
PORT entradaDatos = "", DIR=I, VEC=[31:0]
```

```
PORT salidaDatos = "", DIR=O, VEC=[31:0]
PORT entradaControl = "", DIR=I, VEC=[31:0]
PORT salidaControl = "", DIR=O, VEC=[31:0]

END
```

Cuando modifiquemos el contenido del archivo `mpd`, es necesario reiniciar *XPS*. De lo contrario, el entorno no es consciente de los cambios realizados, y no podremos realizar las conexiones adecuadamente. Tras reiniciar, debemos añadir el hardware al sistema.

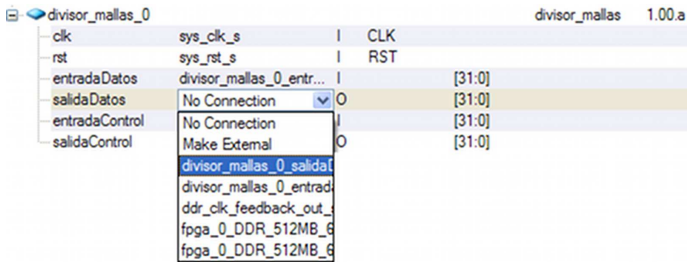
En la parte izquierda del entorno, en la pestaña *IP Catalog* están los dispositivos que podemos añadir al sistema clasificados por categorías. Desplegamos la categoría *Project Repository*, donde veremos que se encuentra el hardware creado. Pulsamos con el botón derecho del ratón sobre el dispositivo que queremos incluir, y seleccionamos la opción *Add IP*:



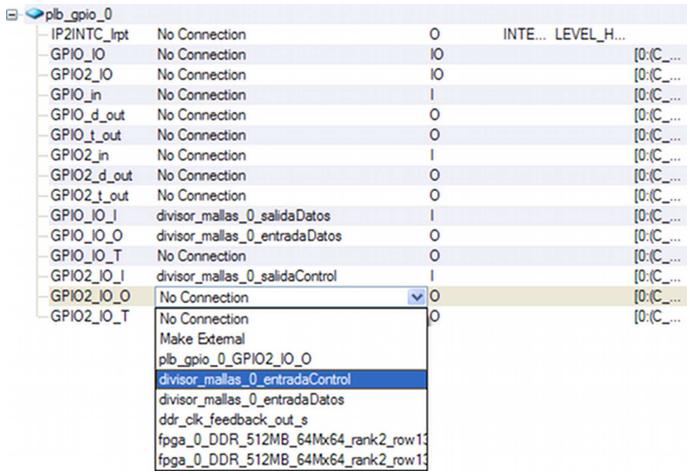
H.3. Conexión de componentes

Tras añadir nuestro hardware, debemos conectar sus señales a las correspondientes del sistema. Para ello, en la parte derecha de la pantalla, en la pestaña *System Assembly View*, seleccionamos la opción *Ports*, con lo que veremos los puertos de los dispositivos. Navegamos hasta nuestro hardware y lo desplegamos:

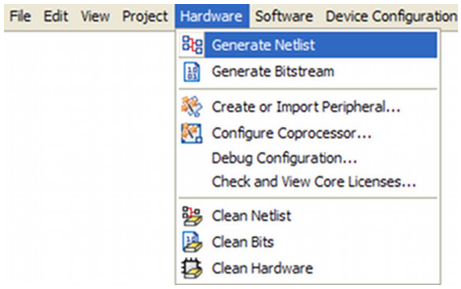
Vemos que están todas las señales sin conectar. Para conectarlas, pulsamos sobre el nombre de la señal que queremos conectar, y aparecerá un menú desplegable. Conectamos todas las señales de ese modo:



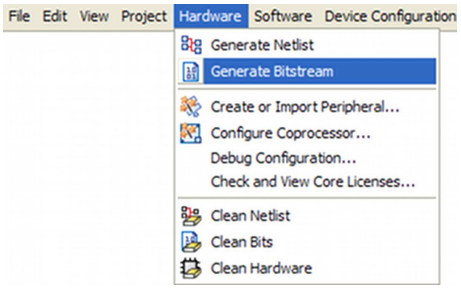
Tras conectarlas adecuadamente, hacemos lo mismo conectando las señales a las entradas y salidas de los GPIO que habíamos añadido al sistema al principio:



Una vez hemos conectado todas las señales, el paso siguiente es crear la *netlist* del sistema. Este es un proceso automático que puede llevar desde unos minutos hasta varias horas, dependiendo de la complejidad del sistema a generar. Para ello, seleccionamos la acción *Generate Netlist* del menú *Hardware*:



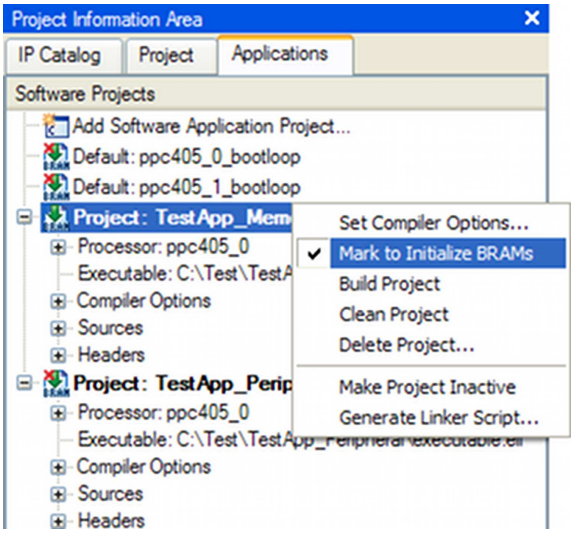
Una vez hayamos creado la *netlist*, es hora de crear el *bitstream*, que es una traducción de la *netlist* a un archivo de configuración de la FPGA. Se realiza seleccionando la opción *Generate Bitstream* del menú *Hardware*:



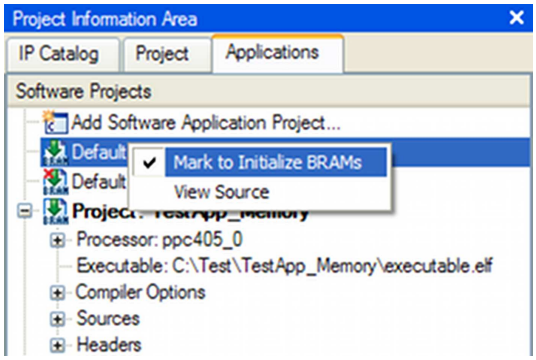
Al igual que en la creación de la *netlist*, la generación del *bitstream* puede llevar desde varios minutos hasta varias horas. En el caso del hardware no optimizado implementado en este proyecto, el tiempo de generación puede llegar a alcanzar las 12 horas en un procesador *AMD Athlon 64 3200+*.

H.4. Programación de la placa

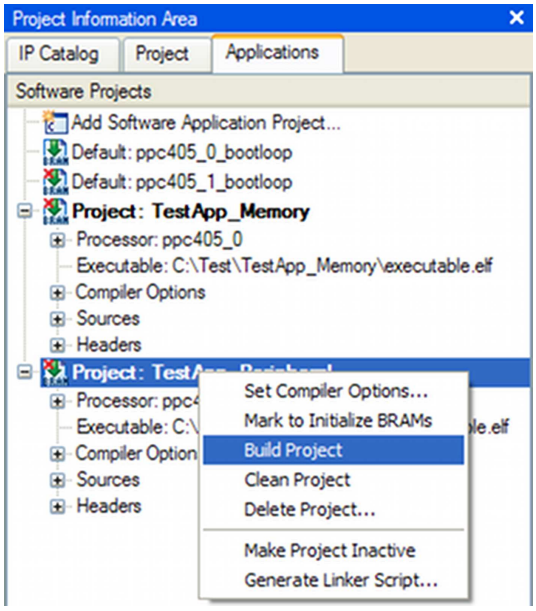
Tras compilar la parte hardware del sistema, hay que integrarla con la parte software. Para ello, lo primero es seleccionar qué programa se cargará inicialmente en la memoria estática de la FPGA. Primero desmarcamos las aplicaciones hardware para que no se utilicen para inicializar el *bitstream*. Tenemos que marcar con qué software queremos inicializar el sistema, en la pestaña *Applications* del *Project Information Area* (parte izquierda). Como vemos en la imagen siguiente, es necesario pulsar con el botón derecho sobre las aplicaciones y desmarcar la opción *Mark to initialize BRAM*:



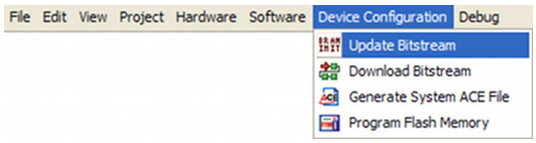
Después, marcamos el *bootloop* del procesador 0 para inicializar el *bitstream*:



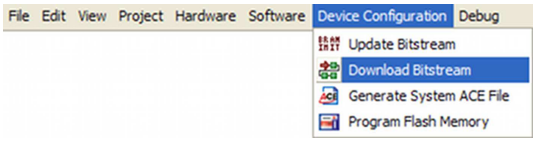
Una vez hemos elegido qué aplicación será parte de la programación inicial del sistema, debemos compilar la aplicación software que ejecutaremos después de programar el sistema. En la parte de *Applications* del *Project Information Area*, pulsamos con el botón derecho del ratón sobre la aplicación que queramos compilar y pulsamos sobre la acción *Build Project*:



Después, actualizamos el *bitstream* con las aplicaciones seleccionadas, ejecutando la acción *Update Bitstream* del menú *Device Configuration*:



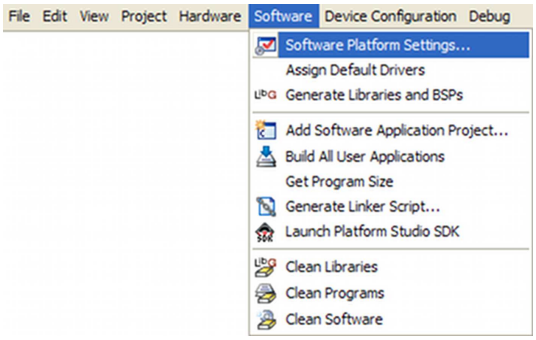
Finalmente, programamos la placa ejecutando la acción *Download Bitstream* del menú *Device Configuration*:



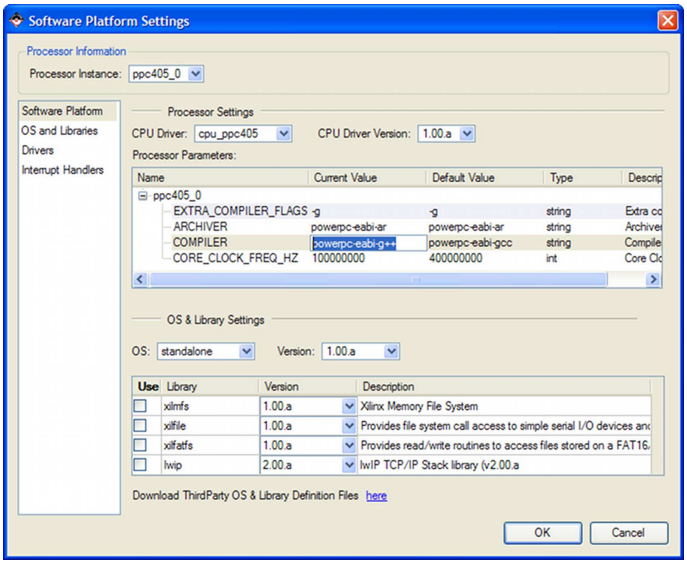
Tras unos segundos, la placa se programa y queda parada ejecutando el *bootloop*. No es necesario indicar al sistema el tipo de cable con el que está conectado la placa; el programa prueba los puertos USB, paralelo, serie, etc. hasta encontrar el adecuado.

H.5. Configuración software

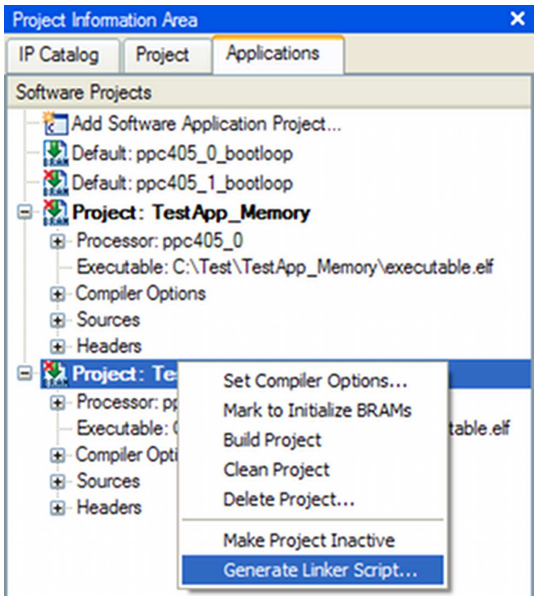
Una vez la parte hardware está terminada, crearemos el programa que se ejecutará sobre la placa. Para empezar, elegiremos el compilador de C++ en lugar del compilador de C que está seleccionado por defecto. Con tal fin, abrimos el diálogo de configuración de la plataforma software: seleccionamos la acción *Software Platform Settings* del menú *Software*, como vemos en la imagen siguiente.



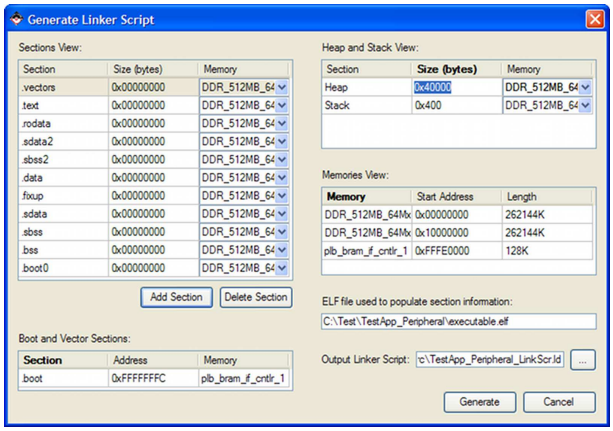
En el diálogo que aparece, cambiamos el nombre del compilador a utilizar, `powerpc-eabi-gcc`, por el nombre del compilador para C++: `powerpc-eabi-g++`. Para realizarlo, escribimos el nombre del compilador nuevo en la columna *Current Value* y pulsamos el botón *Ok*.



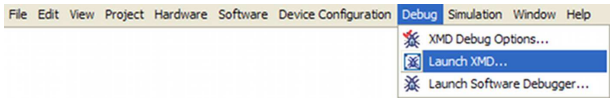
Una vez ajustado el compilador, hay que generar un *script* de enlazado. Este *script* permitirá configurar el tamaño máximo de la pila, el *heap*, los datos y las instrucciones, así como las zonas de memoria en las que se almacenarán dichos datos. Para ello, pulsamos con el botón derecho del ratón sobre el programa que queramos, y seleccionamos la opción *Generate Linker Script*:



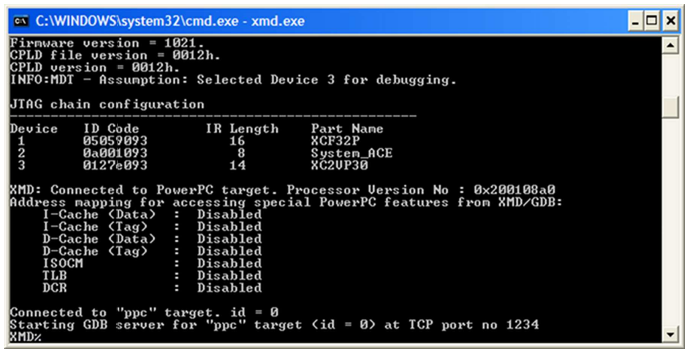
Aparece el siguiente diálogo, en el que tendremos que elegir los valores adecuados. Después, pulsamos el botón *Generate*.



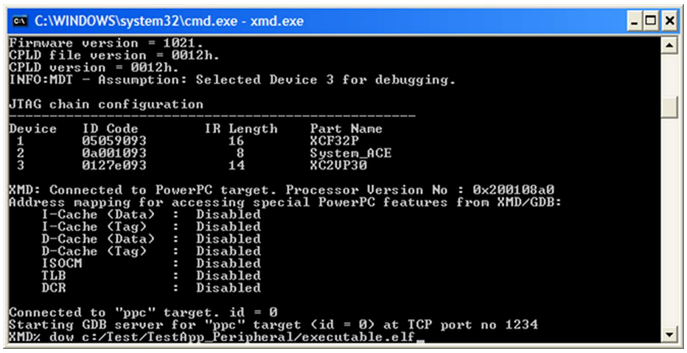
Finalmente, una vez el *script* está generado, hay que volver a compilar como se explicó anteriormente. Al haber configurado el *script* para que el programa cargue las instrucciones y los datos en la memoria DDR, hay que utilizar una utilidad externa para cargar el ejecutable en la placa. Lanzamos la utilidad *XMD*, seleccionando *Launch XMD...* en el menú *Debug*:



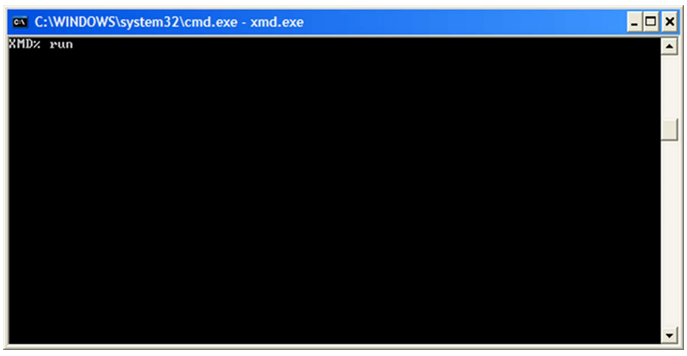
Aparece entonces la ventana de depuración, que se conecta con la placa de forma automática:



Una vez el depurador ha conectado con la placa, descargamos el programa mediante la instrucción `dow`:



Finalmente, ejecutamos la sentencia `run`, con lo que se ejecuta el programa:



Bibliografía

[1] Gerald Farin: *Curves and Surfaces for Computer Aided Design.*

[2] Alan Watt: *3D Computer Graphics.*

[3] Peter Shirley & R. Keith Morley: *Realistic RayTracing*

[4] David A. Patterson & John L. Hennessy: *Computer Organization & Design. The hardware / software interface.*